Marin Fotache (n. 1966) este absolvent al Universităţii „Al.I. Cuza" din Iaşi, Facultatea de Economie şi Administrarea Afacerilor, specializarea Finanţe şi Contabilitate. Este conferenţiar la Catedra de Informatică Economică (www.feaa.uaic.ro). Dintre lucrările publicate : *SQL. Dialecte DB2, Oracle şi Visual FoxPro* (Polirom, Iaşi, 2001) ; este şi autorul a numeroase articole de specialitate publicate în revista *NetReport*.

Ioan (John) Brava (1973-2001) a absolvit Facultatea de Economie şi Administrarea Afacerilor, specializarea Contabilitate şi Informatică de Gestiune (Universitatea „Al.I. Cuza" din Iaşi). Cariera sa universitară (din februarie 2002 urma să avanseze la gradul de lector la Catedra de Informatică Economică) s-a frânt dureros şi neverosimil în octombrie 2001. Aceasta este a doua sa carte (prima se numeşte *Visual Basic. Primii paşi... şi următorii*, Polirom, Iaşi, 2001, scrisă împreună cu Luminiţa Fînaru). Ultima...

Cătălin Strîmbei (n. 1975) a absolvit Facultatea de Economie şi Administrarea Afacerilor, specializarea Contabilitate şi Informatică de Gestiune (Universitatea „Al.I. Cuza" din Iaşi). Din 1998 activează în cadrul Catedrei de Informatică Economică. În colaborare cu Marin Fotache, a scris numeroase articole de specialitate publicate în revista *NetReport*.

Liviu Creţu (n. 1975) a absolvit Facultatea de Economie şi Administrarea Afacerilor, specializarea Contabilitate şi Informatică de Gestiune (Universitatea „Al.I. Cuza" din Iaşi). În perioada martie 1999 - februarie 2001 a fost analist-programator la Societatea de Servicii de Management Feroviar Iaşi, unde a realizat aplicaţia de salarizare a personalului CFR pe zona Moldova. În prezent este asistent şi doctorand la Catedra de Informatică Economică.

© 2002 by Editura POLIROM

www.polirom.ro

Editura POLIROM
Iaşi, B-dul Copou nr. 4, P.O. BOX 266, 6600
Bucureşti, B-dul I.C. Brătianu nr. 6, et. 7, P.O. BOX 1-728, 70700

Descrierea CIP a Bibliotecii Naţionale a României :

Visual FoxPro : ghidul dezvoltării aplicaţiilor profesionale / Marin Fotache, Ioan Brava, Cătălin Strîmbei, Liviu Creţu - Iaşi : Polirom, 2002

592 p., 24 cm

ISBN : 973-683-889-7

I. Fotache, Marin
II. Brava, Ioan
III. Strîmbei, Cătălin
IV. Creţu, Liviu

004.42 VISUAL FOXPRO

Printed in ROMANIA

Marin Fotache
Cătălin Strîmbei

Ioan Brava
Liviu Creţu

VISUAL FOXPRO

GHIDUL DEZVOLTĂRII APLICAȚIILOR PROFESIONALE

130146
B.C.U. - IASI

POLIROM
2002

Conținutul lucrării

Cu siguranță, FoxPro reprezintă cel mai popular subiect românesc în privința bazelor de date. Oarecum paradoxal, dacă ne comparăm cu Vestul. Cât privește **Visual FoxPro**, aici lucrările de referință se numără pe degete. Din păcate. Deoarece nu e vorba numai de modă, cât de funcționalități. În materie de proceduri stocate (în general, așa-numita „logică a bazelor de date") – disponibile altădată doar pe serverele de baze de date, complexe și scumpe –, Visual FoxPro este de ani buni în fața concurenților xBase și chiar a rivalului ce deține gloria SGBD-urilor „personale" – Access. În plus, suportul pentru programarea orientată pe obiecte, lucrul în rețea, ActiveX și Web conferă modernitate și actualitate produsului.

De ce, totuși, VFP nu are gloria internațională pe care o merită? În primul rând, pentru că segmentul de piață căruia i se adresează este tot mai „înghesuit" din mai multe direcții. Pe de o parte, de un Access mult mai ieftin (inclus în Office, nu prea se simte la preț), care, indiscutabil, e mai direct și prietenos cu utilizatorii, mai bine legat de Word, Excel și PowerPoint. Pe de altă parte, cea a SGBD-urilor profesionale – aici lucrurile au evoluat spectaculos și e aproape imposibil de crezut că VFP ar putea spera să atingă gradul de viteză, scalabilitate și securitate din Oracle, DB2, SQL Server, Informix etc. În fine, ultimii ani au popularizat ceea ce noi considerăm actualmente a fi inamicul public numărul 1 al VFP: SGBD-urile de tip *free*. Nu atât MySQL, care, în lipsa procedurilor stocate și cu un nucleu SQL rudimentar, este mai degrabă un raft decât un SGBD (un dBase III plus ceva adăugiri – scuze pentru numeroșii fani ai acestui produs), mai ales PostgreSQL. Simple și puternice, din ce în ce mai sigure – și, să nu uităm, gratuite –, SGBD-urile *free* constituie una dintre puținele soluții pentru dezvoltarea de aplicații, mai ales într-o țară măcinată de jena financiară.

Atunci de ce, totuși, Visual FoxPro? Un prim argument este de ordin istoric. Atâția oameni politici, istorici și alte soiuri de români verzi fac apel la conservarea moștenirii seculare (milenare ș.a.m.d.) a neamului nostru, încât nu ne rămâne decât să schimbăm, scandalos, sistemul de referință și vom vedea că, în materie de FoxPro, avem cea mai importantă „moștenire informatică" (credem că la moștenire se poate adăuga și ceva polimorfism).

Însă, ca și în cazul istoriei, dacă această moștenire nu ne folosește la a fi mai buni și mai performanți *în prezent*, vorbim degeaba. Or, cartea de față ar vrea să arate cum, pornind de la o tradiție informatică de numai o decadă, dispunem de toate facilitățile pentru a obține performanțe informatice la nivelul exigențelor zilelor noastre. Visual FoxPro 6 – care constituie obiectul acestei lucrări –, cu atât mai mult VFP 7, conține toate ingredientele pentru dezvoltarea unor aplicații dezirabile, atât ca funcțiuni, cât și ca preț. Încearcă să vă convingă de asta patru autori care nu s-au limitat doar la a cocheta cu teoria sau catedra universitară.

Capitolul 1 este, în principal, unul de istorie și de „încălzire": descrierea modului de lucru, tipuri de fișiere implicate în proiectele VFP, configurarea sesiunii de lucru. Cu al doilea însă, se intră în miezul bazelor de date: ce înseamnă o bază de date în VFP, care sunt componentele unei BD, ce tipuri de date sunt gestionabile, cum creează asistat tabelele, indecșii, cum se declară regulile de validare la nivel de atribut și înregistrare. Întrucât

constituie, pe de o parte, suportul unui acces rapid la date şi, pe de altă parte, una dintre cele mai frecvente surse de erori ale aplicaţiilor VFP, indecşii au fost trataţi mai pe larg, cu toată deferenţa cuvenită. Firește, nu puteau lipsi din acest capitol chestiunile legate de restricţiile referenţiale.

Al treilea capitol este cel mai xbase-ist, dacă ne permiteţi barbarismul lingvistic. Oricât ar părea de demodată, navigarea prin înregistrări (deplasarea pointerului unei tabele) se regăseşte în toate SGBD-urile profesionale (cei care au folosit cursoare în Oracle, ca să dăm numai un exemplu, ştiu la ce ne referim). VFP rămâne un SGBD care de la dBase se trage. Cu atuurile şi slăbiciunile acestui model de date.

Capitolul 4 parcurge elementele de bază ale programării în VFP: variabile, structuri de control, proceduri, funcţii, tratarea erorilor etc. Una dintre vedetele capitolului este macrosubstituţia, de peste un deceniu motiv de mândrie pentru foxişti şi încă dorită de visualbasicişti (un alt exemplu, la întâmplare).

Următoarele două capitole, 5 şi 6, tratează nucleul SQL implementat în VFP, precum şi extensiile şi substituţiile procedurale ale acestuia. Scopul declarat este, pe de o parte, obţinerea oricărei informaţii dintr-o bază de date (ceea ce duce cu gândul la capitolul 7, dedicat rapoartelor), dar mai ales elaborarea a tot ceea ce înseamnă „logica bazei de date" într-o aplicaţie file-server, client-server sau cum vreţi să-i zicem: funcţii care întorc valori implicite, reguli de validare avansate şi declanşatoare – într-un cuvânt, proceduri stocate.

Al şaptelea capitol este ceva mai relaxant – orice s-ar spune, rapoartele sunt în partea frumoasă şi accesibilă a VFP, pregătind cel mai dur subiect al dezvoltării aplicaţiilor: formularele. Tocmai de aceea, formularele au fost împrăştiate în majoritatea capitolelor următoare. Capitolul 8 este primul care atacă acest subiect, conturând ceea ce este îndeobşte cunoscut sub titulatura de „programare vizuală" (de fapt, în sens riguros, programarea vizuală e un pic diferită): formular, controale, proprietăţi, evenimente şi metode, interesul fiind apoi concentrat pe legătura controalelor cu tabela şi sincronizarea celor două straturi, formular / bază de date.

Capitolul 9 expune câteva aspecte privind utilitatea şi funcţiunile tabelelor derivate, pentru care VFP 6 oferă un suport deplin. Descrierea asistată a creării tabelelor derivate este completată cu „filiera" programatică de declarare a tututor opţiunilor (atribute, reguli de validare, valori implicite, reguli de actualizare a tabelelor de bază etc.) specifice *view*-urilor.

Capitolul 10 reia şi adânceşte problematica formularelor, tratându-se probabil cel mai elegant, dar şi cel mai dificil control VFP – *grid*-ul (reţeaua). Modalităţile de tratare a erorilor apărute într-o sesiune de lucru cu formularele sunt completate cu câteva „finețuri" legate de parametrizare, precum şi utilizarea controalelor de tip ActiveX, ca o prefaţă a ceea ce va fi dezvoltat în capitolul 11 – programarea orientată pe obiecte în VFP 6. În materie de obiecte lucrurile sunt prezentate gradual, cu exemplificări ce s-ar dori edificatoare asupra acestui subiect mereu la modă şi... neînţeles.

Şi capitolul 12 este unul dintre puţinele relaxante prin tematică – meniurile –, deşi discuţia depăşeşte cu mult limitele banalului. În continuare sunt enumerate şi exemplificate cele mai importante comenzi şi funcţii pentru lucrul în medii multiutilizator – mai precis, modul de dezvoltare a aplicaţiilor în arhitecturile specifice VFP – file-server. Iar următoarele două capitole, ce dezvoltă chestiunile arhitecturale, sunt 14 – cu facilităţile VFP privind aplicaţiile Web şi 16 – cu legarea VFP la servere „profesionale" de baze date, în cazul nostru, Oracle.

Între aceste capitole, mai exact, în capitolul 15, se prezintă o tentativă de acoperire a unui gol important al VFP – securitatea. Mecanismul propus, deşi primitiv, se poate dovedi

de mare utilitate în practică, bazându-se pe declanşatoare, tabele de drepturi şi opţiuni de securitate ale sistemului de operare (Windows NT/2000).

Finalul este... executabil. Toate chestiunile prezentate pe parcursul lucrării sunt strânse într-un proiect din care se obţine fişierul executabil sau DLL, care, împreună cu celelalte componente ale unei aplicaţii (baza de date, sistem de asistenţă, imagini etc.), poate fi „împachetat" în kitul de distribuţie. Un paragraf special este alocat unui sistem de asistare (*help*) interactiv, în maniera similară oricărui *help* Windows.

BIBL. CENTR. UNIV. „M. EMINESCU" IAŞI FONDUL ACADEMIC

Acestea ar fi doar câteva repere ale lucrării. Cei care vor avea perseverenţa necesară, vor descoperi destule elemente de detaliu şi, de ce nu, finețe şi vor putea să-şi şlefuiască aplicaţiile VFP pentru a onora orice cerinţă informaţională şi pretenţie managerială (uneori, cele două sintagme sunt disjuncte).

Ar mai fi de adăugat că principalele coduri-sursă (listinguri) din lucrare pot fi descărcate de pe pagina web a Facultăţii de Economie şi Administrarea Afacerilor de la Universitatea „Al.I. Cuza" din Iaşi – http://www.feaa.uaic.ro.

Capitolul 1
Visual FoxPro

Visual FoxPro, produs de firma Microsoft, este un pachet software din categoria sistemelor de gestiune a bazelor de date (SGBD) destinat microcalculatoarelor (există versiuni Windows şi Macintosh).

1.1. Caracterizare generală

Pentru început şi pentru cei neiniţiaţi (iniţiaţii pot sări peste următoarele două-trei paragrafe), vom spune că o bază de date este o colecţie de date de dimensiuni mari, aflate în interdependenţă, memorate pe un suport (de regulă disc magnetic) împreună cu descrierea lor şi a legăturilor dintre ele. Evoluţia nivelului de dezvoltare tehnologică în industria de software a dus la organizarea bazelor de date după mai multe modele, unul foarte răspândit fiind modelul relaţional.

Modelul relaţional a fost pus la punct în anii '70 ai secolului trecut, fiind consacrat în practică în anii '80 şi '90. Acest model organizează datele în *relaţii* sau *tabele*. O relaţie descrie de regulă o clasă de entităţi din lumea reală. Clasa de entităţi este caracterizată prin nişte *atribute* sau *proprietăţi*, ce formează coloanele tabelei (în limbaj curent li se mai spune şi *câmpuri*). Pe liniile tabelei se găsesc *valori* posibile ale atributelor. Ca urmare, fiecare *linie* dintr-o tabelă (căreia i se mai spune şi *înregistrare*) este un *tuplu* de valori ale atributelor şi descrie o entitate particulară. Atributele (coloanele) compun *schema* sau *structura* tabelei (stabilă în timp), iar liniile definesc *conţinutul* acesteia (variabil în timp).

O bază de date relaţională este formată din mai multe tabele care conţin date legate semantic. Legătura se face prin valorile atributelor, fără a se folosi pointeri sau alte tehnici. Spre exemplu, tabela „CLIENŢI" ar conţine date despre clienţii unei firme, în vreme ce tabela „FACTURI" conţine date despre facturile emise clienţilor respectivi. Pentru un client dat, în tabela de facturi pot exista nici una sau mai multe linii, adică facturi emise (avem deci o *relaţie* one-to-many – „unu la mai mulţi"). Pentru ca legătura să fie posibilă, în fiecare dintre cele două tabele trebuie să avem câte un atribut cu aceeaşi semnificaţie (de exemplu, numele clientului ori codul fiscal). De aici şi dubla semnificaţie a termenului „bază de date relaţională": pe de o parte tabelele, sub forma unor relaţii matematice, iar pe de altă parte relaţiile între tabele, sub formă de legături[1]. Şi valorile atributelor trebuie să respecte anumite reguli. Dacă la acestea adăugăm descrierea relaţiilor dintre tabele, obţinem *restricţiile de integritate*. Aceste restricţii, împreună cu numele tabelelor, alcătuiesc *schema bazei de date* şi sunt memorate în *dicţionarul de date* (acesta este de fapt tot o tabelă, dar conţine metadate).

Noţiunile specifice bazelor de date în general şi în Visual FoxPro în particular constituie subiectul capitolului 2.

Un SGBD reprezintă o colecţie de programe care permit definirea structurii unei baze de date şi, ulterior, gestionarea conţinutului acesteia. Prin caracteristicile sale, FoxPro

1. În limba română, cuvântul „relaţie" poate induce o oarecare confuzie. În engleză se folosesc termenii *relation*, respectiv *relationship*.

(atributul de „Visual" s-a adăugat numelui o dată cu versiunile recente pentru Windows) se încadrează în categoria SGBD-urilor relaţionale.

1.2. De la xBase la Visual FoxPro

Conform unei concepţii răspândite printre cei care lucrează în domeniul bazelor de date pentru microcalculatoare, Visual FoxPro aparţine familiei de programe „xBase", numită aşa după „pionierul" *dBase II*, produs al firmei Ashton-Tate.

În 1978, C. Wayne Ratliff (un inginer de aeronave care lucrase pentru NASA, scriind printre altele software pentru navele Viking care au explorat planeta Marte) a pus la punct un mic SGBD dedicat microcalculatoarelor. Microcalculatoarele epocii se remarcau prin preţuri destul de mari şi performanţe extrem de modeste faţă de ceea ce cunoaştem noi astăzi (Ratliff cumpărase cu 1000 de dolari şi montase el însuşi un calculator IMSAI 8080 cu 1 kB de RAM, dar până la urmă, cu toate accesoriile, a cheltuit aproape 6000 de dolari). Programul lui Ratliff se numea *Vulcan* şi ideea de la care pornise în scrierea lui era clasamentul unui campionat de fotbal (american). Ratliff a început să facă reclamă el însuşi produsului său, în revista *BYTE* din octombrie 1979. Curând a avut cereri din partea unor clienţi care vedeau în noul program un instrument ieftin şi comod de ţinere la zi a datelor financiare, contabile, personale etc. În 1980, Ratliff a primit din partea lui George Tate şi Hal Lashlee, patronii unei firme mici, o ofertă de colaborare. Ceva mai târziu, când George Tate conducea casa de software Ashton-Tate[2], produsul lui Ratliff (devenit el însuşi un asociat al firmei) a stat la baza programului *dBase II* (nu a existat o versiune I, în ideea de a sugera maturitatea produsului). Până la începutul anilor 1990, firma Ashton-Tate a vândut un număr mare de copii ale produsului dBase II (program pe 8 biţi, care funcţiona sub sistemul de operare CP/M) şi un număr copleşitor de copii ale produselor dBase III (destinat noilor calculatoare IBM PC pe 16 biţi, cu sistem de operare DOS) şi dBase III+ (cea mai populară versiune dBase). Curând a apărut şi concurenţa, sub chipul unor firme ca Borland (astăzi Inprise), Nantucket, Microrim, Computer Associates, MicroPro şi Fox Software. Produsele tuturor acestor firme erau asemănătoare ca arhitectură cu dBase (clone ale acestuia), de unde şi reunirea lor sub apelativul „xBase". Din păcate, versiunea dBase IV 1.0 din 1988 a fost un insucces răsunător, din cauza frecventelor erori cu care produsul era „împănat". În loc de a depana erorile versiunii pentru DOS, programatorii de la Ashton-Tate au fost îndrumaţi să înceapă lucrul la o versiune pe 32 de biţi, pentru sistemul de operare OS/2 al IBM, care, şi el, întârzia să apară. Firma Fox Software, cu produsul *FoxBase+*, o clonă dBase, a profitat de descumpănirea ivită pe piaţă, umplând rapid golul. În urma acestor evenimente, Ashton-Tate, slăbită financiar, a fost cumpărată de Borland, care a perfecţionat produsul dBase pentru a-l face să ruleze sub Windows. Dar nici aici succesul nu a fost cel scontat, deoarece Microsoft lansase între timp SGBD *Access*, care nu era poate la fel de performant ca dBase, dar se bucura de o interfaţă prietenoasă şi de o mare uşurinţă în utilizare, fiind proiectat să fie exploatat direct în logica interfeţei grafice utilizator[3] specifică Windows. Totuşi Borland a creat versiuni îmbunătăţite ale programului dBase.

2. Nu exista nici o persoană numită Ashton în firmă, dar Tate a considerat că numele „cu liniuţă" ar suna mai bine în reclame.
3. GUI – *Graphical User Interface*.

Fox Software şi-a perfecţionat produsul, care a devenit FoxPro. Versiunea 2.0 a fost unul dintre marile succese comerciale ale anului 1991. La acea vreme, Microsoft încerca să impună Access, dar acesta era oarecum imatur. În lume exista un parc deloc neglijabil de microcalculatoare mai vechi, care nu puteau rula Windows (deci nici Access), dar care erau foarte potrivite pentru FoxPro. În plus, aplicaţiile FoxPro se remarcau printr-o viteză de execuţie mare, impresionantă faţă de concurentul dBase[4]. Microsoft a achiziţionat în cele din urmă firma Fox Software (1992). Din momentul achiziţiei, producţia a continuat cu versiunile 2.5 şi 2.6 (pentru DOS, Windows, Macintosh şi UNIX), apoi cu versiunile „Visual" 3, 5 şi 6. Dacă versiunea 3.0 a fost una de tranziţie, fiind lansată independent, versiunile 5 şi 6 erau parte a unui pachet software numit Visual Studio. La data când scriem aceste rânduri, versiunea 7.0 a produsului Visual FoxPro este încă în stadiul Beta2.

Versiunile 2.x ale produsului FoxPro au oferit, pe lângă motorul bazei de date, facilităţi pentru scrierea şi execuţia propriilor programe, proiectarea asistată a interfeţelor (meniuri, formulare sau machete de preluare a datelor – în mod caracter, tipic mediului DOS), proiectarea de rapoarte, caracteristici specifice mediilor de dezvoltare a aplicaţiilor. Versiunile „Visual" au introdus o serie de facilităţi sistematizate în tabelul 1.1.

Tabelul 1.1. Versiuni Visual FoxPro

An apariţie	*Versiune*	*Caracteristici noi*
1995	Visual FoxPro 3.0	Versiune pe 16 biţi (destinată mediilor DOS + Windows 3.x), funcţionează şi cu Windows 95 sau NT. Gestionarul de aplicaţii (`Project Manager`) a fost reproiectat. Suport pentru conceptul de bază de date şi restricţiile de integritate aferente (introduce dicţionarul de date). În versiunile anterioare se lucra cu tabele independente. Posibilitatea de a crea proceduri stocate şi declanşatoare. Limbaj îmbogăţit cu tehnici de programare orientată-obiect. Orientare spre paradigma *event-driven* proprie aplicaţiilor Windows. Mod de lucru grafic. Instrumente vizuale de proiectare a interfeţelor (formulare şi controale), printre care şi asistenţii `Form Wizard`, `Graph Wizard`, `Pivot Table Wizard`, `One-To-Many Report Wizard`. Se pot crea biblioteci de clase proprii. Suport îmbunătăţit pentru lucrul în reţea: mecanismul *buffering*, tranzacţiile (maximum 5 niveluri de imbricare a tranzacţiilor).
1996	Visual FoxPro 5.0	Versiune îmbunătăţită, optimizată pentru 32 de biţi, a produsului. Cerinţe hardware mai reduse ca la versiunea 3, cerinţe software Windows 95 sau Windows NT. Performanţe sporite în manipularea datelor. Creşterea compatibilităţii cu alte baze de date ce respectă standardul `ANSI SQL`. Formatele de fişiere noi fac imposibilă revenirea la versiunea 3.0 după conversia aplicaţiilor în formatul 5.0. Posibilitatea folosirii de controale `ActiveX`, pentru o funcţionalitate sporită a aplicaţiei.

4. Viteza mare era datorată unor tehnici de indexare şi căutare performante, reunite sub numele *Rushmore*.

An apariție	*Versiune*	*Caracteristici noi*
		Mai multă flexibilitate: structura bazei de date, ca și procedurile stocate, pot fi modificate „din mers", fără a fi necesară deschiderea exclusivă a bazei de date. Facilități de conectare la servere de baze de date (standardul ODBC). Instrumente puternice de depanare a aplicațiilor. *Wizard* pentru crearea de aplicații executabile (și biblioteci DLL) și instalarea lor asistată.
1998	Visual FoxPro 6.0	Câteva zeci de comenzi și funcții noi față de versiunea 5. Se îmbogățește colecția de evenimente. Mecanism îmbunătățit pentru gestionarea proiectelor (`Project Hook Manager`) și a bazei de date. Utilitar de creare a claselor și componentelor îmbunătățit. Instrumente pentru optimizarea execuției programelor (`Coverage Profiler`). Bibliotecă de clase predefinite (`FFC - FoxPro Foundation Classes`). Suport pentru crearea de aplicații Web. Suport pentru `Microsoft Transaction Server`, un produs strategic în viziunea Microsoft. Utilitar de verificare a datelor calendaristice, pentru corecția „problemei mileniului".
2001	Visual FoxPro 7.0	Peste 50 de comenzi, funcții și variabile de sistem noi. Suport pentru programare distribuită (`COM - Component Object Model`). Tehnologii `IntelliSense` și `AutoList Members` încorporate în editorul de programe-sursă: sintaxa unei funcții ori comenzi, lista de proprietăți a unui obiect etc. apar la scrierea funcției ori la scrierea numelui obiectului și apăsarea caracterului „selector" (punct). Facilități îmbunătățite de navigare în codul-sursă. Compatibilitate sporită cu `Microsoft SQL Server`. Driver `OLE DB` ce permite crearea de aplicații client/server non-FoxPro, cu baze de date FoxPro. Câteva funcții `XML (eXtended Markup Language)`. Evenimente pentru care pot fi definite metode la nivelul *container*-ului bazei de date.

Aceste caracteristici au făcut ca FoxPro să evolueze de la un SGBD simplu, cu oarecare facilități de realizare a aplicațiilor, la un adevărat *mediu integrat de dezvoltare*. În accepțiune clasică, un mediu de dezvoltare (în engleză ADE – *Application Development Environment*, iar în franceză AGL – *Atélier de Génie Logiciel*) constă dintr-un editor de texte pentru scrierea codului-sursă într-un limbaj dat, un compilator pentru limbajul respectiv și câteva facilități de natura depanatoarelor, pentru verificarea aplicației. O dată cu perfecționările în domeniul hardware-ului și cu creșterea atât a numărului de limbaje și de variante ale acestora, cât și a numărului de dezvoltatori de aplicații, aceste instrumente s-au diversificat și îmbunătățit, integrând noi facilități.

Astăzi, printr-un *mediu integrat de dezvoltare* (IDE – *Interactive Development Environment*) înțelegem un *cadru* completat de un set larg de *instrumente* pentru realizarea

aplicațiilor. *Instrumentele* sunt mai ales cele din categoria *generatoarelor de aplicații*, în care utilizatorul construiește aplicația mai mult prin specificații declarative decât prin programare procedurală. De exemplu, machetele de introducere a datelor sunt proiectate vizual, programatorul plasând pe un ecran (*formular*) diferite obiecte (*controale*) predefinite, fiecare „știind" să interpreteze anumite evenimente – ca, de pildă, un click de mouse ori o apăsare de tastă. Nu e nevoie a scrie instrucțiuni detaliate care să asigure desenarea acelor obiecte de pe ecran la momentul execuției; generarea codului-sursă, pornind de la specificațiile create, cade în sarcina instrumentului respectiv. Cât privește *cadrul* de dezvoltare a aplicației, acesta constă în elemente specifice fiecărei categorii de aplicații. În cazul particular al aplicațiilor cu baze de date, elementele sunt de natura modulelor de cod, meniurilor, formularelor, rapoartelor, bibliotecilor de obiecte sau de proceduri și funcții. Pentru gestionarea lor facilă, acestea sunt organizate în *proiecte* ce constituie punctul de plecare pentru aplicații complete. Evident, dintr-un astfel de mediu integrat nu lipsesc uneltele de depanare și optimizare a execuției aplicației. Dacă inițial editorul, compilatorul și depanatorul se utilizau separat, mediile integrate contemporane înglobează toate instrumentele în cadrul unui același pachet software, a cărui instalare, configurare și folosire nu ridică probleme deosebite.

În lumea PC-urilor, firma Borland International s-a remarcat încă din anii '80 ca promotor al mediilor integrate, prin produsele sale din categoria *Turbo* (Turbo Pascal, Turbo Prolog și Turbo C). O dată cu dezvoltarea interfeței Windows (și, mai târziu, a sistemului de operare Windows 95/98/NT/2000), Microsoft a introdus mediile de dezvoltare din categoria *Visual* (*Visual Basic*, *Visual C++*, *Visual FoxPro*, *Visual J++*), iar Borland – mediile *Visual dBase*, *Delphi* (Pascal orientat-obiect), *C++ Builder*. Am dat doar aceste exemple, întrucât produsele firmelor Microsoft și Borland cunosc cea mai mare răspândire pe piață.

În general, un mediu de dezvoltare integrat presupune ca „aport" în sistemele de aplicații client-server partea „de client" sau *front-end*, prin care înțelegem, în mare, interfața-utilizator. Partea „de server" (*back-end*), respectiv definirea bazei de date și logicii[5] acesteia, intră într-o altă sferă, realizându-se cu instrumente livrate o dată cu SGBD-ul respectiv. Privit din această perspectivă, sistemul Visual FoxPro dezvăluie o natură duală: pe de o parte constituie un SGBD relațional, iar pe de altă parte, integrează instrumentele caracteristice unui mediu de tip *Rapid Application Development* (RAD). În consecință, proiectele dezvoltate în mediul Visual FoxPro au în vedere atât partea de modelare a bazei de date și de definire a procedurilor stocate, cât și crearea obiectelor aplicației care utilizează respectiva bază de date. De menționat că limbajul și structurile de control specifice bazei de date sunt similare celorlalte obiecte ale aplicației. Se obține astfel fie o aplicație *stand-alone* (în care toată „munca", de la gestionarea interfeței la gestiunea bazei de date, revine unui singur sistem de calcul), fie, în condițiile unei separări accentuate a interfeței-utilizator de baza de date, doar partea de client a unei aplicații client-server tradiționale (*two-tier* – în două straturi). În acest ultim caz, baza de date și logica acesteia sunt implementate pe un server de baze de date (uzual Microsoft SQL Server sau Oracle). Fabricantul oferă chiar un asistent (*Upsizing Wizard*) pentru portarea unor aplicații FoxPro către platforme mai puternice. Aplicațiile în care baza de date este implementată în Visual FoxPro iar interfața-utilizator în alt mediu de programare sunt de asemenea realizabile.

5. Prin *logică* se înțeleg aici regulile de funcționare a mediului real pentru care se implementează baza de date (așa-zisele *business rules*). De exemplu: „Pentru a fi titularul unui permis de conducere trebuie să ai minimum 18 ani".

FoxPro-ul de astăzi mai păstrează din familia xBase doar modul de organizare a datelor (o tabelă înseamnă un fişier pe disc) şi limbajul de programare aferent (care, la rândul lui, a fost substanţial îmbogăţit). Caracteristicile sale fac din el un mediu de dezvoltare rapid, robust, puternic, relativ uşor de învăţat. Dacă mai adăugăm la aceasta şi răspândirea pe care aplicaţiile FoxPro (nu neapărat în varianta *Visual*, dar mai devreme sau mai târziu sună ceasul migrării) o cunosc în întreprinderile româneşti, raţiunea scrierii acestui material se conturează ceva mai bine.

1.3. Moduri de lucru în FoxPro

Încă de la versiunile „fragede", FoxPro a permis lucrul în mai multe moduri, după preferinţele şi abilităţile utilizatorului. Astfel, cel mai simplu este *modul asistat* sau *modul de lucru meniu*, în care utilizatorul are la dispoziţie opţiuni de meniu pentru operaţiile dorite. *Modul direct* presupune cunoaşterea sintaxei comenzilor FoxPro, comenzi scrise în *fereastra de comenzi*, unde tasta `Enter` are rolul de a le lansa în execuţie.

Sintaxa (formatul general) unei comenzi FoxPro arată astfel:

```
VERB [<lista_expresii>] [<domeniu>]
     [FOR <expL1>] [WHILE<expL1>]
   [TO PRINTER | TO ARRAY < tablou>
             | TO <variabila_memorie>
           | TO FILE <fisier>]
```

Pentru cei nefamiliarizaţi cu notaţia Backus-Naur (metalimbaj), furnizăm câteva explicaţii:

- componentele scrise cu majuscule sunt cuvinte rezervate; se folosesc exact în aceeaşi formă, eventual prescurtate la primele 4 caractere;
- componentele scrise cu minuscule, în paranteze unghiulare – construcţii ale utilizatorului (nume de tabele, nume de câmpuri, expresii matematice ori logice etc.);
- [ ] desemnează o construcţie opţională;
- { } desemnează mai multe construcţii din care una este obligatorie;
- | separă două construcţii care se exclud reciproc;
- ... desemnează construcţii eliptice (care se repetă).

În afara comenzilor FoxPro native, Visual FoxPro dispune de un nucleu `SQL` ale cărui comenzi sunt foarte aproape de specificaţiile `ANSI SQL`.

În figura 1.1 s-a exemplificat modul de execuţie a unei comenzi FoxPro dintre cele mai simple, „?", care are rolul de a afişa ceva direct pe ecranul FoxPro.

Multe dintre comenzile accesibile din meniuri se scriu automat în fereastra de comenzi în momentul acţionării lor, ceea ce este, credem, o bună ocazie pentru a le deprinde pe cele mai frecvent folosite, fără a fi nevoie să „tocim" formatele lor.

În fine, *modul de lucru program* se referă la scrierea de programe-sursă cu ajutorul editorului specializat. Într-un program se pot folosi aproape toate comenzile modului de lucru direct, plus câteva comenzi specifice reprezentând structurile de control. O dată scris,

programul se memorează ca fişier, se compilează (opţional[6]) şi se execută cu ajutorul comenzii `DO <nume_program>`. Cu modul de lucru program ne vom (re)întâlni în capitolul 4.

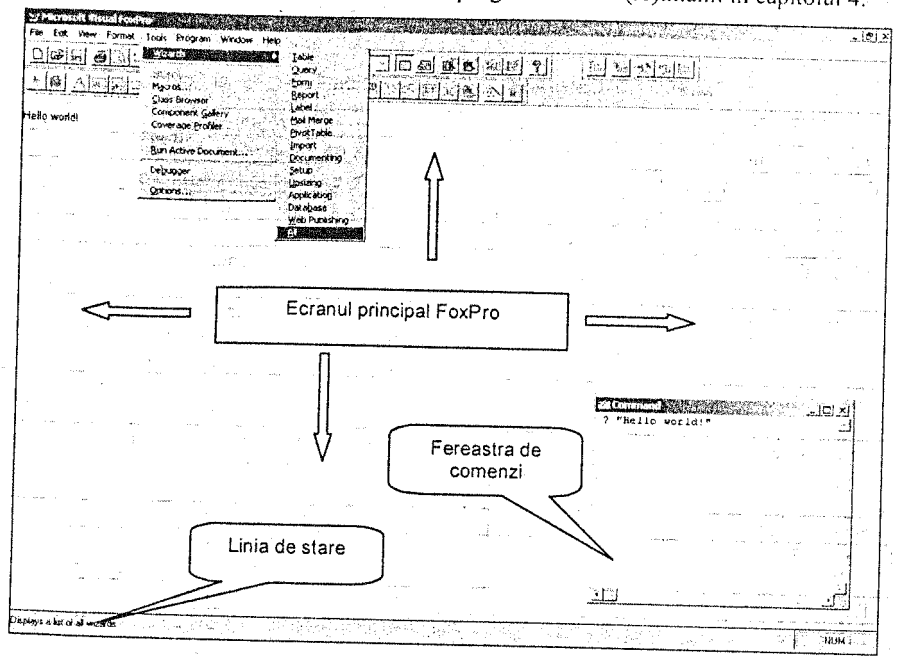


Figura 1.1. Ecranul principal Visual FoxPro

1.4. Crearea unui proiect Visual FoxPro

O etapă extrem de importantă din ciclul de viaţă al unui sistem informatic este faza implementării. Acum programatorii trebuie să transpună într-un limbaj de programare toate specificaţiile formulate în faza de proiectare.

În general, elaborarea unei aplicaţii într-un mediu integrat de dezvoltare constă în construirea unui proiect care trebuie să gestioneze o serie de componente cum ar fi:

- structurile de date utilizate de aplicaţie, cum ar fi schemele de baze de date;
- componente care vor genera interfaţa aplicaţiei: meniuri, formulare, rapoarte;
- componente dedicate exclusiv structurării logicii de prelucrare specifice aplicaţiei: module de cod (proceduri);
- diverse biblioteci refolosibile (clase), utile în standardizarea aplicaţiilor şi creşterea productivităţii echipei de dezvoltare.

6. La lansarea în execuţie, FoxPro compilează automat programul dacă versiunea-sursă este mai recentă decât eventuala versiune compilată.

Aceste componente se concretizează într-un mare număr de fişiere memorate pe suportul specific (disc), iar aici FoxPro nu face excepţie, tipurile de fişiere necesare într-o aplicaţie fiind multe şi diferite, după cum se vede şi în tabelul 1.2.

Tabelul 1.2. Câteva tipuri de fişiere folosite de Visual FoxPro

Tip fişier	*Extensie*	*Explicaţii*
DataBase Container	`.dbc`	Dicţionarul de date (tabelă cu metadate)
Database Container Text	`.dct`	Tabelă memo[7] asociată dicţionarului de date
Database Container indeX	`.dcx`	Fişierul index asociat dicţionarului de date
DataBase File	`.dbf`	Tabele ale bazei de date
InDeX file	`.idx`	Fişier index asociat unei tabele (index simplu, stocat ca fişier distinct)
Compound inDeX file	`.cdx`	Fişier index asociat unei tabele (index compus, adică mai mulţi indecşi sunt memoraţi în acelaşi fişier) Indecşii compuşi sunt preferabili celor simpli
FoxPro Text	`.fpt`	Fişier asociat tabelelor care conţin câmpuri de tip *memo*
Screen File (Form)	`.scx`	Fişier ce conţine definiţiile realizate prin intermediul constructorului de machete de ecran (forms = formulare)
Screen Memo	`.sct`	Fişier memo asociat unui formular
Report File	`.frx`	Fişier ce conţine informaţii despre datele incluse într-un raport (listă) şi definirea parametrilor săi
Report Memo	`.frt`	Fişier memo asociat unui raport
Label File	`.lbx`	Fişier ce conţine informaţii despre datele incluse într-o etichetă, împreună cu definirea parametrilor săi (etichetele sunt liste mai mici, imprimabile mai multe pe o pagină)
Label Memo	`.lbt`	Fişier memo asociat unei etichete
Menu File	`.mnx`	Fişier ce conţine definiţiile realizate prin intermediul constructorului de meniuri
Menu Memo	`.mnt`	Fişier memo asociat unui meniu
Generated Menu Program	`.mpr`	Programul-sursă generat pornind de la macheta meniului
Compiled Menu Program	`.mpx`	Programul compilat rezultat din fişierul `.mpr`
Generated Query Program	`.qpr`	Programul-sursă tip filtru sau interogare, generat de constructorul de interogări
Compiled Query Program	`.qpx`	Programul compilat tip filtru (interogare)
Project File	`.pjx`	Proiect (conţine date despre baza de date, formularele, rapoartele, meniurile, programele unei aplicaţii)
Project Memo	`.pjt`	Fişier memo asociat unui proiect
FoxPro Program	`.prg`	Program-sursă FoxPro
Compiled FoxPro Program	`.fxp`	Program compilat (pornind de la `.prg`)
Compilation Error File	`.err`	Fişier cu lista erorilor de la compilarea unui program-sursă

7. Fişierele memo sunt un element caracteristic mediului FoxPro şi conţin date nestructurate, de mari dimensiuni, legate de alte tabele cu date. Litera „T" într-o extensie de fişier semnifică fişier memo.

Tip fişier	*Extensie*	*Explicaţii*
Executable Program	`.exe`	Fişier executabil obţinut dintr-un proiect ce conţine măcar un program-sursă, un formular ori un meniu. Este versiunea finală a unei aplicaţii. Funcţionează şi fără a avea mediul FoxPro instalat pe calculatorul-ţintă, dar numai împreună cu câteva biblioteci FoxPro
FoxPro Visual Class Library	`.vcx`	Bibliotecă de clase FoxPro (create cu `Class Wizard`)
FoxPro Visual Class Library Memo	`.vct`	Fişier memo asociat unei biblioteci `.vcx`
FoxPro dynamic Link Library	`.fll`	Bibliotecă de funcţii de nivel scăzut (`API`)
Dynamic Link Library	`.dll`	Bibliotecă de funcţii Windows de uz general

Este evident că aproape orice obiect dintr-o aplicaţie Visual FoxPro are nevoie de două sau chiar mai multe fişiere. Modul de organizare a diferitelor categorii de componente într-un proiect Visual FoxPro este ilustrat în figura 1.2.

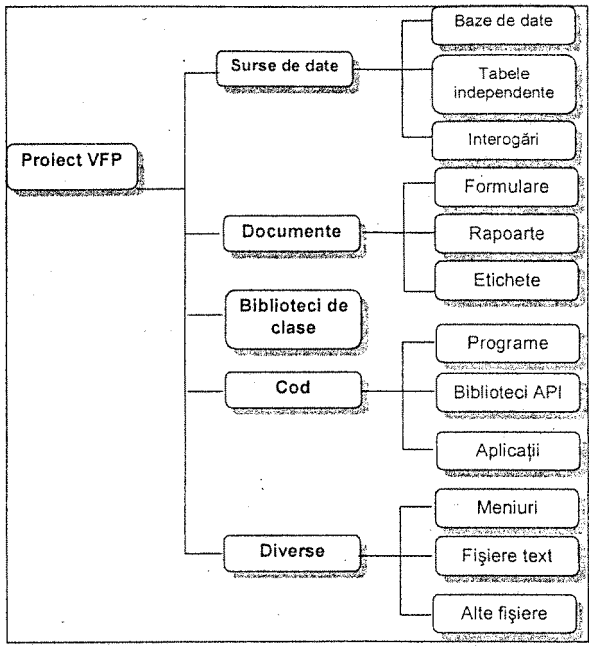


Figura 1.2. Elementele unui proiect Visual FoxPro

Instrumentul specific Visual FoxPro pentru gestionarea fişierelor care formează un proiect este *Project Manager* (figura 1.3), care se lansează prin opţiunile de meniu `File →New...→Project` sau în mod direct prin comanda `CREATE PROJECT <nume_proiect>`.

Structura unui proiect este redată fie arborescent, elementele specifice proiectului putând fi accesate din cadrul de pagină `All`, fie sub forma altor cadre de pagină, fiecare

categorie de elemente fiind posibil de explorat în pagina specifică ei: `Data`, `Docs` sau `Documents`, `Classes`, `Code`, `Other` (aceste pagini sunt utile în „segmentarea" proiectelor mari).

Fereastra `Project Manager` prezintă în partea dreaptă o serie de butoane prin care se lansează rapid anumite acțiuni specifice gestiunii proiectului (`New`, `Add`, `Remove`, `Build`...) sau care au incidență asupra unui obiect specific al proiectului:

- pentru baze de date – `Open`, `Close`, `Modify`;
- pentru tabele – `Browse`, `Modify`;
- pentru formulare – `Modify`, `Run`;
- pentru rapoarte – `Modify`, `Preview`, `Run`;
- pentru programe, meniuri – `Modify`, `Run`;
- pentru clase – `Modify`.

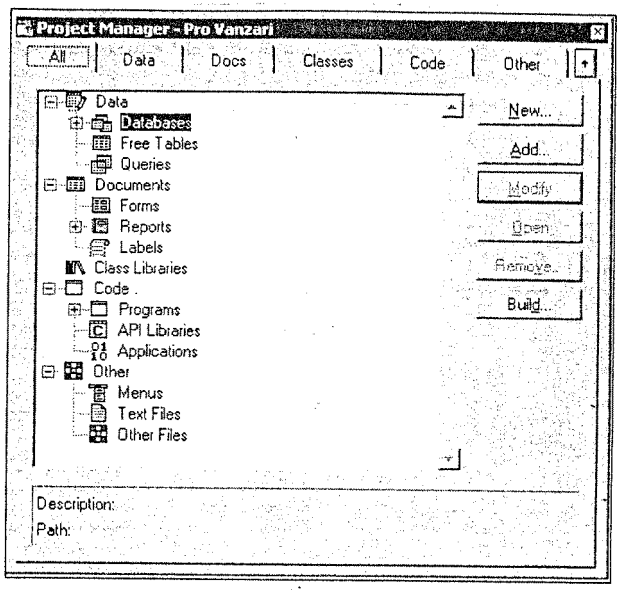


Figura 1.3. Fereastra instrumentului `Project Manager`

Textele afișate pe aceste butoane depind de elementul selectat în arborele din stânga, iar acționarea lor scutește utilizatorul de căutarea comenzii adecvate în meniuri ori de scrierea sa manuală în fereastra de comenzi.

1.5. Configurarea mediului Visual FoxPro

Înainte de a trece la explicarea și exemplificarea oricăror concepte din Visual FoxPro, vom prezenta câteva comenzi zise „de configurare", în care Visual FoxPro pur și simplu abundă. Zisa abundență, pe lângă ușoara confuzie pe care o induce utilizatorului începător, are și părțile sale bune, în sensul că oferă o mare flexibilitate mediului de lucru.

1.5.1. Comenzile SET

Un prim format al acestor comenzi este :

```
SET <parametru> TO [<valoare>]
```

Un alt format, foarte răspândit, este:

```
SET <parametru> ON|OFF
```

Sunt și formate care combină cele două opțiuni, respectiv:

```
SET <parametru> ON|OFF|TO [<valoare>]
```

Stările multora dintre parametrii modificați de aceste comenzi (vezi tabelul 1.3) se pot afla prin invocarea funcțiilor din categoria `SYS(număr)`.

Tabelul 1.3. Cele mai folosite comenzi de tip `SET`

Parametru	*Tip*	*Semnificație*
`ANSI`	`ON|OFF`	Dacă este `ON`, două șiruri de caractere vor fi comparate caracter cu caracter, deci „Ion" va fi diferit de „Ion_ "; în caz contrar, sunt posibile „pseudoegalități" de tip „Tom" = „Tommy".
`BELL`	`ON|OFF` sau `TO <numefisier>`	Activează/dezactivează semnalizarea sonoră (producerea sunetului are loc cu comanda `??CHR(7)`). Opțiunea `TO` specifică un fișier `.wav` pe post de sunet de atenționare. De exemplu: `SET BELL TO „c:\<windows\media\ding"`.
`CARRY`	`ON|OFF`	Dacă este `ON`, o înregistrare nou-adăugată într-o tabelă prin comanda `APPEND` (vezi capitolul 3) va copia în câmpurile sale valorile din înregistrarea precedentă.
`CENTURY`	`ON|OFF|` `TO [<secol>` `[ROLLOVER` `<an>]]`	Activează sau dezactivează afișarea anului în format cu 4 cifre (este utilă la introducerea datelor calendaristice din secole diferite). Formatul cu `TO <secol>` permite specificarea secolului implicit, iar opțiunea `ROLLOVER <an>` permite, în cazul introducerii doar ultimelor două cifre, să se decidă de la ce an inclusiv începe secolul trecut și la care an se termină secolul curent[8]. Valoarea implicită a parametrului `<an>` este anul curent minus 50.
`CLOCK`	`ON|OFF|STATUS`	Afișează sau nu ora exactă în colțul din dreapta-sus al ecranului ori pe bara de stare, în colțul din dreapta-jos.
`DATABASE`	`TO <nume` `-baza-de-date>`	Specifică numele unei baze de date care va deveni curentă. Această bază de date trebuie să fie deschisă.
`DATE`	`TO AMERICAN |` `ANSI | BRITISH` `| FRENCH |` `GERMAN` `| ITALIAN |` `JAPAN | TAIWAN` `| USA | MDY |` `DMY | YMD`	Permite stabilirea unui format de afișare a datelor calendaristice. Formatul implicit este cel `AMERICAN`, `LL/ZZ/AA`, dar se poate opta pentru altele (`BRITISH` ori `FRENCH` sunt de forma `ZZ/LL/AA`, `GERMAN` este `ZZ.LL.AA`, `ANSI` este `AA.LL.ZZ` etc.). Numărul de cifre afișat pentru an depinde de starea parametrului `CENTURY` (vezi mai

8. De exemplu, dacă anul curent este 2001 și argumentul *an* are valoarea 51, introducerea valorilor de la 51 la 99 semnifică anii 1951...1999, pe când valorile 00...50 vor însemna 2000...2050.

Parametru	*Tip*	*Semnificaţie*
	| SHORT | LONG	sus). Formatele SHORT şi LONG preiau modul de afişare stabilit în Windows Control Panel şi nu ţin seama de comanda SET CENTURY.
DECIMALS	TO <numar>	Specifică numărul de zecimale (între 0 şi 18) afişat la expresiile numerice.
DEFAULT	TO <cale>	Specifică directorul de lucru (cel în care Visual FoxPro caută orice bază de date, tabelă, program, formular, raport etc. când se lansează o comandă de deschidere ori execuţie). Iniţial, acesta este directorul în care s-a instalat Visual FoxPro[9]. Parametrul cale poate fi: o unitate de disc, o unitate urmată de o cale de directoare, un director-copil (situat în ierarhie „sub" cel existent), una din notaţiile prescurtate \ sau . . Calea stabilită prin această comandă se află prin combinaţia de funcţii SYS(5) + SYS(2003).
DELETED	ON|OFF	Cu ON, înregistrările marcate pentru ştergere într-o tabelă vor fi ignorate de comenzile de prelucrare a înregistrărilor (excepţii notabile sunt INDEX, REINDEX şi SELECT - SQL). Cu OFF, aceste înregistrări sunt „ascunse". Viteza de prelucrare a datelor din tabele mari se reduce dacă parametrul DELETED este în starea ON.
ESCAPE	ON|OFF	Permite întreruperea (starea ON) execuţiei unui program sau a unei comenzi prin apăsarea tastei Esc.
EXCLUSIVE	ON|OFF	Când este ON, utilizatorul curent încearcă să obţină acces exclusiv la deschiderea unei baze de date sau tabele; când este OFF, forma de acces este cea partajată (*shared*). Unele comenzi (în special cele de întreţinere a bazei de date: INDEX, PACK, MODIFY STRUCTURE, VALIDATE DATABASE etc.) cer acces exclusiv. De folosirea judicioasă a acestui parametru depinde succesul unor aplicaţii multiutilizator.
EXACT	ON|OFF	Permite compararea exactă (ON) sau nu a două şiruri de caractere cu lungimi diferite („popescu" şi „pop"). Compararea are loc până se termină şirul din membrul drept, dacă starea acestui parametru este OFF, deci rezultatele obţinute pot fi neaşteptate. În loc de comanda SET EXACT ON, se poate folosi în comparaţiile şirurilor de caractere operatorul == (doi de „egal").
FDOW	TO <cifră>	Precizează prima zi a săptămânii (implicit este 1, adică duminică). Are efect în anumite calcule cu date calendaristice, folosind funcţia DOW (în ce zi a săptămânii va cădea 25 decembrie 2001?).
FIELDS	ON | OFF | LOCAL | GLOBAL	Permite o „decupare pe verticală" a tabelei, precizând care dintre câmpuri pot fi accesate (dacă

9. De regulă, c:\Program Files\Microsoft Visual Studio\VFP98.

Parametru	*Tip*	*Semnificaţie*
		este ON, trebuie urmată de SET FIELDS TO <listă-de-câmpuri>).
FILTER	TO <expresie-logică>	Stabileşte un criteriu pentru „decuparea pe orizontală" a tabelei, permiţând accesarea doar a liniilor care satisfac expresia. Filtrul este ignorat de comanda SELECT - SQL.
FUNCTION <număr> sau FUNCTION <nume>	TO <expresie>	Permite atribuirea unei expresii apăsării unei taste funcţionale (exemplu: SET FUNCTION 2 TO „Salut" sau SET FUNCTION CTRL+F2 TO „Ciao!"). Efectul acestei asignări de taste se anulează cu comanda CLEAR MACROS.
MULTILOCKS	ON|OFF	Permite (ON) accesul concurent la înregistrările unei tabele. Este necesar pentru implementarea mecanismului *buffering* (vezi capitolul 13).
ORDER	TO TAG <nume-index>	Permite specificarea unui index existent într-un fişier index de structură ca şi index principal al tabelei (pentru indecşi vezi capitolul 2). Indexul principal influenţează operaţiile de căutare (cu comanda SEEK).
PATH	TO <listă-căi>	Permite specificarea uneia sau mai multor căi (separate prin virgulă), în afara căii stabilite prin SET DEFAULT în care Visual FoxPro să caute fişiere. Convenţiile de denumire sunt aceleaşi ca la SET DEFAULT. De exemplu: SET DEFAULT TO k:\aplicatie SET PATH TO forms,progs,imagini\foto
PRINTER	ON|OFF|TO <nume-imprimantă>	Direcţionează către imprimantă rezultatele unor comenzi. Opţiunea TO permite alegerea altei imprimante decât cea implicită. Încercaţi în fereastra de comenzi următoarea linie de cod: SET PRINTER TO GETPRINTER()
SAFETY	ON|OFF	Dacă este ON, utilizatorul este avertizat înainte de operaţiile care suprascriu un fişier existent (tabelă, index, fişier text etc.). Dacă este OFF, operaţia are loc fără avertisment. Folosiţi această comandă cu precauţie.
STATUS BAR	ON|OFF	Afişează/ascunde linia de stare a ecranului principal Visual FoxPro.
STRICTDATE	TO 0|1	Are rolul de a asigura compatibilitatea cu anul 2000. TO 0, permite atribuiri de genul: datacalend={21/12/2001} TO 1, cere o atribuire neambiguă, de tipul: datacalend={^2001-12-21}
SYSMENU	ON|OFF|AUTOMATIC |TO [*listaoptiuni*] | TO [DEFAULT]	Are rolul de a ascunde (OFF) ori afişa (ON) un meniu, care poate fi meniul principal Visual FoxPro ori anumite opţiuni specificate într-o listă. Numele opţiunilor de meniu încep cu _M şi continuă cu textul lor: _MFILE, _MTOOLS etc.; mai multe detalii în capitolul 12.
TALK	ON|OFF	Decide dacă mesaje despre rezultatele execuţiei unor comenzi vor fi afişate sau nu pe ecran ori pe bara de

Parametru	*Tip*	*Semnificaţie*
		stare (un exemplu este cel al comenzii `INDEX`, care va afişa „*n* records indexed").
`UDFPARMS`	`TO VALUE | REFERENCE`	Stabileşte modul de transmitere a parametrilor de la o procedură apelantă către o procedură apelată (pentru detalii, vezi capitolul 4).

Există şi o comandă `SET` fără nici un parametru, al cărei rol este de a afişa la un moment dat tabelele deschise, împreună cu zonele lor de lucru şi cu eventualii indecşi (vezi figura 1.4). Despre tabele, zone de lucru şi indecşi veţi afla mai multe în capitolele următoare. Deocamdată precizăm că această comandă este similară ca efect opţiunii de meniu `Window→Data Session`.

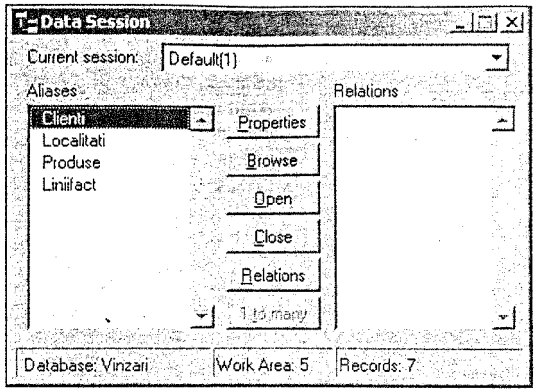


Figura 1.4. Efectul comenzii `SET`

Mulţi dintre parametrii comenzilor `SET` pot fi modificaţi şi de o manieră asistată, prin dialogul lansat de opţiunea de meniu `Tools→Options`. În figurile 1.5...1.7 sunt exemplificaţi câţiva asemenea parametri.

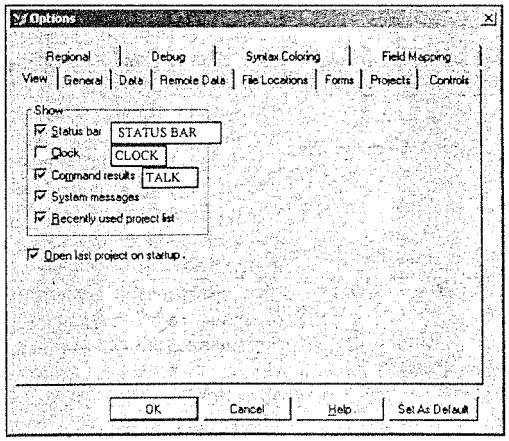


Figura 1.5. Parametri ai mediului Visual FoxPro (1)

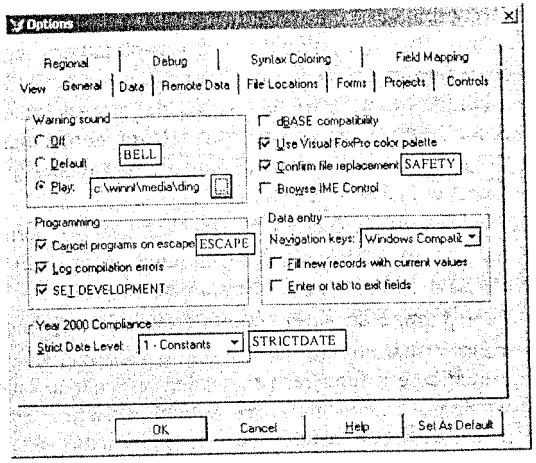


Figura 1.6. Parametri ai mediului Visual FoxPro (2)

Notă. Situaţia multora dintre parametri poate fi urmărită şi prin scrierea comenzii `DISPLAY STATUS` în fereastra `Command`.

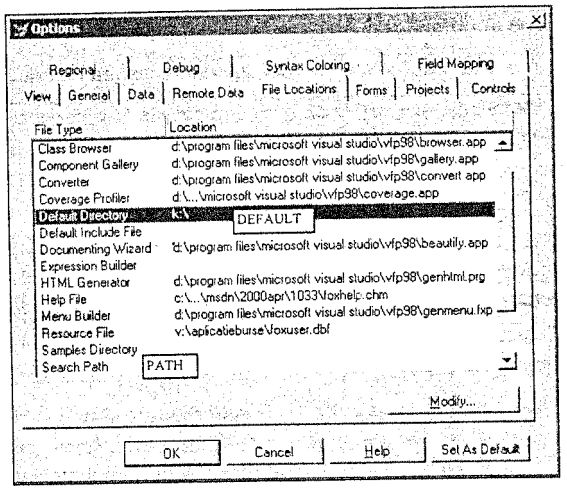


Figura 1.7. Parametri ai mediului Visual FoxPro (3)

1.5.2. Comenzile ON

În afara comenzilor `SET`, o serie de opţiuni de configurare sunt disponibile prin comenzi care încep cu cuvântul-cheie `ON`. Iată câteva:

- `ON KEY LABEL` – permite asignarea unei funcţionalităţi oarecare unei taste sau combinaţii de taste.

Exemplu: combinația `Ctrl+F10` are ca efect în Visual FoxPro maximizarea ferestrei curente. Putem „deturna" acest comportament scriind în fereastra de comenzi (de exemplu) `ON KEY LABEL CTRL+F10 ?"Salut"`, prin care, la apăsarea numitei combinații se va afișa pe ecranul principal Visual FoxPro cuvântul „Salut". Restaurarea comportamentului „normal" se face cu `ON KEY LABEL CTRL+F10`. Dacă sunt mai multe asemenea definiții în uz și vrem să le anulăm pe toate, vom folosi `ON KEY`.

- `ON ESCAPE` – asemănătoare cu `ON KEY LABEL` - permite atribuirea unei misiuni tastei `Esc`, după întreruperea programului curent (deci funcționează numai dacă parametrul `SET ESCAPE` este `ON`).
- `ON SHUTDOWN` – permite declanșarea unei comenzi sau proceduri atunci când se încearcă ieșirea din Visual FoxPro, cu una dintre modalitățile `QUIT`, `File→Exit` sau butonul de închidere a ferestrei principale, ori chiar la încercarea de a părăsi mediul Windows fără a fi închis mai întâi sesiunea Visual FoxPro. Este un bun prilej de a atenționa un utilizator asupra necesității/iminenței unei salvări/pierderi a datelor. De remarcat că părăsirea mediului Visual FoxPro nu mai are loc decât dacă programatorul o dorește în mod explicit.

De regulă, acestor comenzi li se atribuie nu o simplă instrucțiune, ci o procedură întreagă invocată prin comanda `DO`. Pentru edificare, dăm totuși un exemplu. Astfel, în fereastra de comenzi se poate scrie: `ON SHUTDOWN DO atentie`

Redăm aici codul procedurii „atenție", scrisă și salvată separat[10] (caracterele `&&` introduc comentarii pe marginea codului).

Listing 1.1. Procedura „Atentie" declanșată la ieșirea din Visual FoxPro

```
IF 6=MESSAGEBOX("Sunteti sigur(a)?")  && daca se raspunde "Yes" la aceasta intrebare
    QUIT    && se paraseste Visual FoxPro, altfel ramanem pe loc
ENDIF
```

Într-o aplicație „îngrijită", comanda `ON SHUTDOWN DO...` se va plasa într-un program principal, iar în procedura de terminare a aplicației (vom reveni la multe dintre aceste lucruri în capitolele următoare) se va scrie o instrucțiune `ON SHUTDOWN` fără nimic altceva, pentru a restaura comportamentul „normal" al mediului Visual FoxPro.

1.5.3. Variabile de mediu

Pentru configurarea numeroaselor sale opțiuni, Visual FoxPro dispune, încă de la versiunile cele mai vechi, de o mulțime de variabile de mediu (*environment variables*) ale căror caracteristici comune sunt acelea de stabilire a valorilor unor parametri și faptul că numele tuturor încep cu caracterul „liniuță-de-subliniere" (*underscore*). Vom da aici câteva exemple:

- `_DBLCLICK` – stabilește intervalul dintre două click-uri succesive pentru a fi considerate dublu-click. Acest interval e măsurat în fracțiuni de aproximativ 1/18 s;
- `_DIARYDATE` – conține data curentă, dar i se poate atribui și o altă dată;

10. După cum se va vedea în capitolul 4.

- `_PAGENO` – conține numărul paginii curente, fiind utilizată la imprimarea rapoartelor (vezi capitolul 7);
- `_SCREEN` – este o referință la ecran (fereastra principală Visual FoxPro). Se comportă ca un formular special (vezi capitolul 17), având multe din proprietățile și metodele acestuia. Astfel, o comandă precum `_SCREEN.BackColor = RGB(255,0,0)` va colora fundalul ecranului în roșu pur[11]; `_SCREEN.Caption = „Sesiune de lucru Visual FoxPro"` va afișa acest text în bara de titlu a ferestrei principale Visual FoxPro;
- `_TALLY` – va conține numărul de înregistrări prelucrate de ultima comandă, fiind utilă la compunerea unor mesaje de stare adresate utilizatorului, în unele reguli de validare sau declanșatoare etc.
- `_THROTTLE` – permite specificarea unei pauze (de la 0 la 5,5 s) între execuția liniilor succesive dintr-un program și se utilizează numai la depanarea programelor;
- `_VFP` – variabilă ce permite accesul la o serie de proprietăți ale *instanței* Visual FoxPro (programului executabil), cu precizarea că la un moment dat, pe un singur calculator se pot executa mai multe sesiuni de lucru Visual FoxPro. De exemplu, `?_VFP.Name` va afișa textul „Microsoft Visual FoxPro". Cu comanda `?_VFP.FullName` putem afișa pe ecranul principal calea completă a programului `VFP6.EXE`, iar cu comanda `?_VFP.Version` se va afișa versiunea curentă (6.0).

În încheiere, precizăm că parametrii stabiliți prin comenzile enumerate mai sus sunt memorați în baza de date a sistemului (`Registry`) – vezi figura 1.8. Alte opțiuni, cum ar fi pozițiile diferitelor ferestre, bare de instrumente ori meniuri pe ecran sunt memorate în fișiere specifice zise „fișiere de resurse" (`foxuser.dbf`).

1.5.4. Fișierul CONFIG.FPW

Încă de la „vârste fragede", FoxPro a permis memorarea unor parametri specifici, precum și efectuarea unor operații preliminare deschiderii sesiunii de lucru cu ajutorul unui fișier special de configurare. Pentru versiunile „Visual", acest fișier se numește `CONFIG.FPW` și conține o serie de comenzi de parametrizare. Este un fișier de tip text, ce poate fi scris cu un editor simplu, gen *Notepad*. Un exemplu este dat în listingul 1.2.

Listing 1.2. Exemplu de fișier de configurare

```
DEFAULT=„c:\acasa"
PATH=„progs, reports, database"
CLOCK=STATUS
TALK=OFF
EXCLUSIVE=OFF
EXACT=ON
COMMAND=_SCREEN.Caption = „VFP, configuratie de lucru pentru acasa"
```

11. Cu funcția RGB se poate compune o culoare din trei valori: pentru roșu (*Red*), verde (*Green*) și albastru (*Blue*). Valorile variază, fiecare, de la 0 (absență) la 255 (saturație maximă).

Câteva explicații:

- liniile de forma `<parametru>=<valoare>` țin locul comenzilor obișnuite `SET <parametru>=<valoare>`;
- liniile de tip `COMMAND=...` reprezintă comenzi care, în mod normal, s-ar scrie în fereastra `Command` (în exemplul de față, textul „VFP, configurație de lucru pentru acasă" va fi plasat în bara de titlu a ferestrei principale Visual FoxPro).

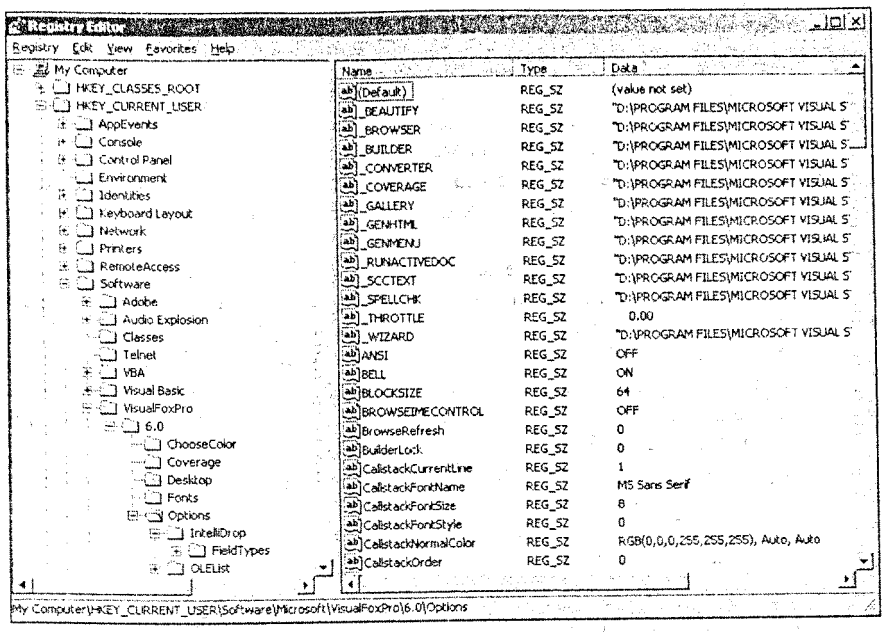


Figura 1.8. Locația din `Registry` care memorează parametrii de lucru

Necesitatea acestui fișier apare datorită setărilor diferite care sunt cerute pentru funcționarea unor programe create de utilizator. Am văzut că parametrii sunt memorați în `Registry`. De acord, numai că aici intervine ceva adesea trecut cu vederea, chiar de către cei experimentați: un program Visual FoxPro compilat ca fișier `.EXE` *nu* consultă baza de date `Registry`, deci fie se va folosi un program principal („punct de intrare" în aplicație) ce cuprinde toate comenzile `SET` necesare (sau, pentru programatorii cu experiență, funcții pentru citirea valorilor din `Registry`), fie se va folosi un fișier `CONFIG.FPW`, fie (mai rar) o soluție mixtă.

Cum se folosește acest fișier, `CONFIG.FPW` ? Soluția constă în crearea unui *shortcut* la programul principal (`VFP6.EXE` sau altul, în cazul unei aplicații compilate). Comanda asociată acestui *shortcut* va conține în cale și numele programului executabil, urmate de un spațiu și de textul `-C<cale-și-nume-fișier-de-configurare>` (vezi figura 1.9). Lansarea unei sesiuni Visual FoxPro cu parametrii stabiliți prin acest fișier se va face, evident, prin dublu-click pe *shortcut*-ul respectiv.

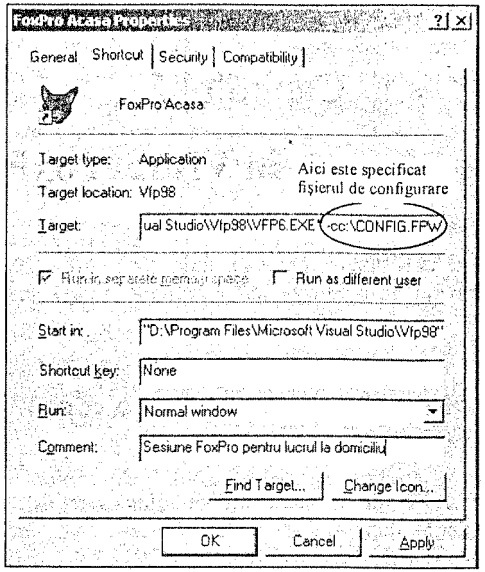


Figura 1.9. *Shortcut* care folosește un fișier `CONFIG.FPW`

Dacă dezvoltatorul dorește să poată lansa mai multe sesiuni, fiecare cu parametrii săi, atunci poate recurge la mai multe *shortcut*-uri, fiecare folosind un alt fișier `CONFIG.FPW`, plasat într-un director distinct și indicat la crearea *shortcut*-ului, cum se arată în figura 1.9.

Listing 1.3. Alt exemplu de fișier de configurare

```
DEFAULT=„k:\aplicatie"
PATH=„programe, rapoarte, baza_de_date"
TALK=ON
EXCLUSIVE=OFF
EXACT=ON
COMMAND=_SCREEN.Caption = „VFP, configuratie de lucru la serviciu"
```

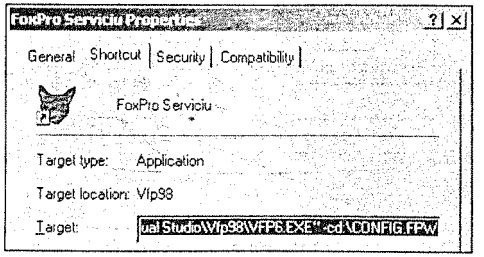


Figura 1.10. Alt *shortcut*, alt fișier de configurare...

Capitolul 2

Baze de date în Visual FoxPro

Noţiunea de *bază de date* apare la finele anilor '60. Anii '70 şi chiar o parte din deceniul 8 plasează termenul în umbra *băncilor de date*. Astăzi, această confuzie a cedat locul altora.

2.1. Conceptul de „bază de date" şi o confuzie de pe vremea xBase-urilor

La vremea apariţiei bazelor de date, aplicaţii cu volum mare de date erau organizate în jurul fişierelor independente, denumite arhitecturi *file-based* sau *flat files*. Necazul era că descrierea unui asemenea fişier (chiar în aplicaţii medii existau zeci de asemenea fişiere de date) trebuia făcută în fiecare program care-l utiliza. Iar în condiţiile în care într-o aplicaţie există sute de asemenea programe (fiecare sortare, actualizare, listare are nevoie de propriile programe ce conţin, de aproape fiecare dată, descrierea datelor din fişiere), vă închipuiţi ce efort presupune modificarea tipului sau lungimii unui câmp, în general tot ce ţine de modificarea structurii de date.

La modul cel mai simplu şi simplist, o bază de date conţine fişierele de date propriu-zise, plus un fişier special în care se stochează structura fişierelor de date şi legăturile dintre acestea, precum şi o serii de restricţii ce pot fi definite asupra câmpurilor şi înregistrărilor din fişiere – vezi figura 2.1.

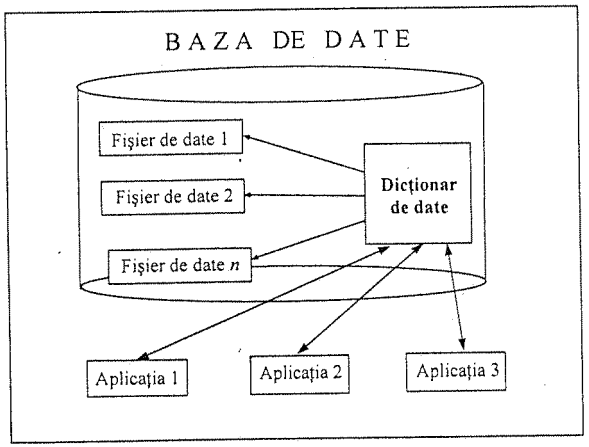


Figura 2.1. Schemă de principiu a unei baze de date

Nu vom intra în detalii cu privire la alte modele de organizare a datelor decât cel relaţional. Se cuvine însă de remarcat că în lumea xBase, din care au făcut parte „înainte-mergătorul" dBase, alături de RBase, FoxPro, Clipper şi alte produse mai puţin cunoscute, termenul SGBDR (Sistem de Gestiune a Bazelor de Date Relaţionale) a fost folosit cu oarecare îndrăzneală, pe alocuri nejustificată.

Unul dintre motive ţine de faptul că, până la jumătatea anilor '90, xBase-urile etichetau ca *bază de date* orice tabelă – fişier `.DBF`. Şi acum mai întâlnim lucrări în care se vorbeşte despre comanda dBase/FoxPro `CREATE` prin care „se creează baza de date CLIENTI", ceea ce nu-i tocmai în regulă. Drept este că orice `.DBF` (tabelă) are propriul *header*, în care se stochează structura fişierului. Însă dicţionarul de date, numit în Visual FoxPro *containerul bazei de date*, este cu mult mai generos, în sensul înglobării nu numai a tabelelor, ci şi a restricţiilor bazei, a definiţiilor tabelelor virtuale (*view*-urilor), a funcţiilor şi procedurilor stocate, a conexiunilor cu surse externe (gestionate de alte SGBD-uri), a tabelelor derivate la distanţă etc. Pe disc, containerul bazei se găseşte sub forma a trei fişiere cu extensiile `.DBC`, `.DCT` şi `.DCX`, din care primul este containerul propriu-zis (celelalte două sunt un fişier memo şi un index asociate `.DBC`-ului).

Principalele tipuri de obiecte ale unei baze date sunt vizibile din modul de afişare în cadrul managerului de proiect – figura 2.2.

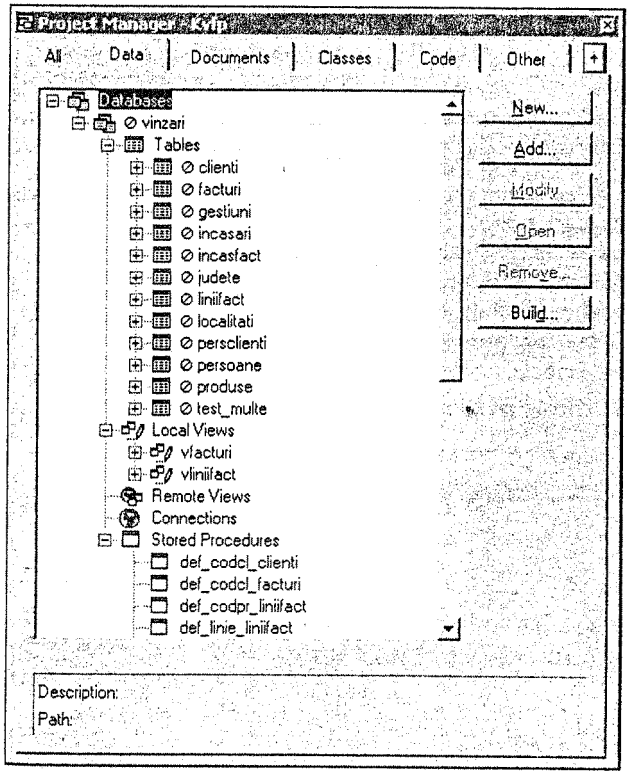


Figura 2.2. Obiecte din schema bazei de date `VINZARI`

Baza de date VINZARI este compusă din:

- **tabelele**: CLIENTI, FACTURI, GESTIUNI etc.;
- **tabelele virtuale** (*view*-uri) **locale:** VFACTURI, VLINIIFACT etc.;
- o serie de **proceduri stocate** (funcţii, proceduri, declanşatoare) dintre care primele patru sunt vizibile în figură (`def_codcl_clienti`, `def_codcl_facturi`...).

Crearea unei baze date se poate realiza în oricare dintre cele trei moduri prezentate în capitolul 1: din fereastra de comenzi, printr-un program în care apare comanda CREATE DATABASE, însă cel mai lejer este să se aleagă opţiunea New din cadrul de pagină Data al proiectului, ca în figura 2.3.

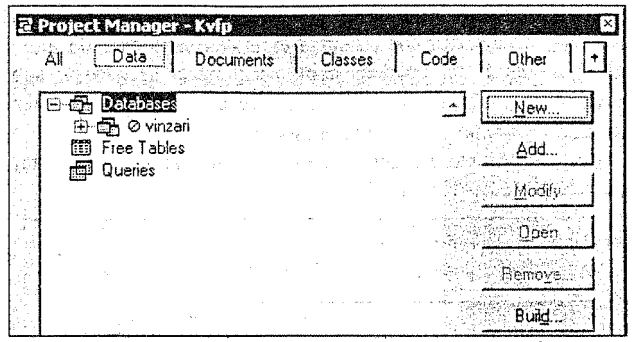


Figura 2.3. Vizualizarea bazei de date în cadrul proiectului

Instrumentul ideal pentru lucrul asistat cu o bază de date este *Proiectantul bazei* (*Database Designer* – figura 2.4), deoarece permite, în egală măsură:

- adăugarea de tabele (şi tabele virtuale) la cele existente;
- modificarea structurii tabelelor;
- vizualizarea conţinutului tabelelor;
- stabilirea de relaţii permanente între tabele (pe baza indecşilor);
- ştergerea de tabele, tabele derivate, relaţii permanente etc.

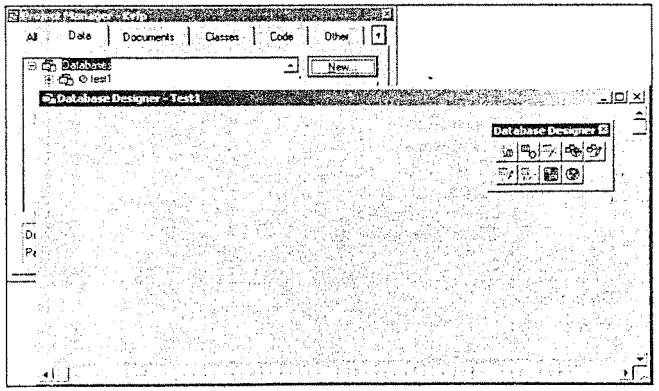


Figura 2.4. Modulul de proiectare a bazei de date

Modulul de proiectare a bazei poate fi lansat fie din cadrul proiectului (după poziţionarea pe numele bazei de date şi apoi alegerea opţiunii Modify), fie direct, prin comanda MODIFY DATABASE.

Ulterior creării, bazei de date îi este actualizat conţinutul şi/sau modificată structura, în conformitate cu tranzacţiile şi procesele pe care le reflectă şi cu noile nevoi informaţionale. Utilizarea unei bazei de date presupune în prealabil deschiderea sa, operaţiune care se poate realiza:

- prin meniu;
- din proiect;
- prin comanda OPEN DATABASE.

Pe parcursul celorlalte capitole vor fi tratate fiecare dintre tipurile de obiecte ale bazei de date. Pentru început însă, ne vom concentra atenţia pe noţiunea centrală de gestiune a informaţiilor – **tabela**.

2.2. Tabelele unei baze de date în Visual FoxPro

În termeni pretenţioşi, tabela este unitatea organizaţională fundamentală dintr-o bază de date relaţională. O bază de date conţine una sau mai multe tabele, în funcţie de complexitatea aplicaţiei. Chiar la aplicaţii de anvergură medie, numărul tabelelor poate atinge ordinul zecilor, iar VFP se descurcă onorabil în aceste situaţii.

Gruparea atributelor în tabele, relaţiile dintre tabele, numărul tabelelor dintr-o bază de date sunt chestiuni ce privesc procesul de proiectare a bazei, proces care îl include pe cel de normalizare. Lucrarea de faţă nu-şi propune să intre în detalii „normalizatoare", ci vom pleca de la o structură gata „fabricată", pe care o vom prezenta ceva mai jos.

Este recomandabil ca toate tabelele să fie incluse în baza de date, chiar dacă legarea lor de alte tabele nu pare întotdeauna atât de evidentă. Cu toate acestea, tabelele pot exista şi independent de orice bază de date. Acestea sunt denumite *libere* (*free tables*) şi, contrar denumirii, sunt supuse unor restricţii severe: nu pot avea nume lungi, nu li se pot defini restricţii la nivel de linii, declanşatoare etc. Astfel încât, vom „pedala" doar pe tabele ce aparţin bazei de date, aşa „dependente" cum sunt.

2.2.1. Tipuri de date în VFP

Înainte însă de a prezenta structura tabelelor bazei de date „martor", să zăbovim preţ de câteva rânduri asupra tipologiei datelor gestionabile în VFP. Orice tabelă este alcătuită din *atribute* sau *câmpuri*. La creare, fiecărui atribut i se precizează numele, tipul, lungimea şi, eventual, numărul de poziţii fracţionare (pentru câmpuri de tip numeric).

În funcţie de semnificaţie, orice atribut poate fi de tip:

- **integer** – conţine numere întregi; ex.: 34580; conform documentaţiei VFP6, intervalul în care trebuie să se încadreze o valoare de tip integer este cuprins între -2147483647 şi 2147483647;
- **numeric** – număr real cu sau fără poziţii fracţionare; ex.: 34580, 456.657789; valoarea minimă acceptată pentru acest tip de dată este - .9999999999E+19, iar valoarea maximă 9999999999E+20;
- **float** – similar tipului numeric;

- **double** – număr real reprezentat în virgulă mobilă dublă precizie (pentru calcule ştiinţifice); intervalul de reprezentare a datelor de tip `double` este cuprins între +/-4.94065645841247E-324 şi +/-8.9884656743115E307;
- **currency** – numere în format monetar; la partea fracţionară pot fi maximum patru poziţii; la conversia din formatul numeric, atunci când sunt cinci sau mai multe poziţii zecimale, se face trunchierea (cu rotunjire); ex.: 456.6578, 54. Valoarea minimă reprezentabilă în acest format este - 922337203685477.5807, iar cea maximă 922337203685477.5807 (ne pare rău că nu putem despărţi în silabe un număr !);
- **character** – şiruri de maximum 254 de caractere; ex.: „ana are mere.", 'Ana are mere!';
- **date** – dată calendaristică, stocabilă/afişabilă de formatul ales (ISO, american, francez, german, englez etc.); constantele de tip dată calendaristică sunt scrise între acolade, folosindu-se şi accentul circumflex pentru indicarea anului; ex: {^2000/12/31};
- **datetime** – data plus ora; ex.: {^2001/07/23 14:55:56} care este echivalentă cu {^2001/07/23 2:55:56PM}[12];
- **logical** – poate avea două valori logice, `true` (.T.) sau `false` (.F.);
- **memo** – este util pentru stocarea unor blocuri de text mai mari de 254 de caractere, stocare exterioară tabelei, într-un fişier cu extensia `.FPT` (ex.: pentru CV-uri, procese-verbale etc.); lungimea unui câmp `memo` este limitată numai de spaţiul disponibil pe disc;
- **general** – constituie o referinţă către un obiect OLE stocat într-un fişier extern bazei (foaie de calcul, document Word etc.).

Într-un program sau în fereastra de comenzi, pentru a determina tipul unei variabile sau câmp se poate folosi funcţia `TYPE()`. De exemplu,

```
? TYPE('{^2001/07/23 2:55:56PM}')
```

afişează `T` (de la `DateTime`).

Literele returnate de funcţia `TYPE()` sunt următoarele:

- `C` – pentru tipul character,
- `N` – pentru numeric, integer, float şi double,
- `D` – date,
- `T` – datetime,
- `Y` – currency,
- `L` – logical,
- `M` – memo şi
- `G` – general.

12. Acest format „straniu" pentru constante de tip `Date` şi `DateTime` a fost introdus de Visual FoxPro 6.0 pentru a evita confuziile legate de anul 2000. Versiunile mai vechi foloseau formatele {LL/ZZ/AA} ori {ZZ/LL/AA}, format încă operabil în Visual FoxPro 6.0 dacă se foloseşte comanda `SET STRICTDATE TO 0` (zero).

2.2.2. Conversia între tipurile de date

Uneori, în aplicaţii este necesară conversia între diversele tipuri de date, cel mai adesea pentru concatenări. În paragraful dedicat indecşilor, vom vedea că, atunci când cheia de indexare este compusă, VFP solicită concatenarea atributelor, posibilă numai cu ajutorul conversiei atributelor în şiruri de caractere.

Conversia în şiruri de caractere necesită următoarele funcţii:

- pentru datele numerice, funcţia `STR()`; ex.: `STR(NrFact,7)`, unde 7 este lungimea şirului proaspăt obţinut;
- pentru date calendaristice, funcţiile `DTOS()` sau `DTOC()`; ex.: `?DTOS({^2001/07/23})` afişează `20010723`, în timp ce `DTOC({^2001/07/23})`, `23/07/2001` (atunci când formatul datei este `british`);
- pentru atribute, variabile sau constante dată/timp se foloseşte funcţia `TTOC()`;
- datele logice (boolean) nu pot fi convertite direct în şiruri, dar se pot folosi, în expresii, `IF`-uri imediate (`IIF`-uri).

Relativ frecvent sunt folosite şi funcţiile: `VAL()` pentru conversia unui şir de caractere într-un număr, `CTOD()` – care transformă un şir de caractere în dată calendaristică, `TTOD()` – ce întoarce data calendaristică dintr-o constantă/variabilă/atribut de tip dată/timp şi `CTOT()` – ce transformă un şir de caractere într-o valoare de tip dată/timp.

2.2.3. Valori nule

Unul dintre cele mai discutate ingrediente ale modelului relaţional îl reprezintă teoria (şi practica) valorilor nule. O valoare nulă este o valoare lipsă sau inaplicabilă. Dificultatea este cu atât mai mare, întrucât până la versiunea Visual FoxPro 3, foxiştii nu erau obişnuiţi cu asemenea gen de valori, zero sau spaţiu ţinând locul valorilor necunoscute sau neaplicabile.

O tabelă BENEFICIARI ar putea conţine câte o linie pentru fiecare persoană fizică şi juridică care a cumpărat un produs/serviciu de la firma noastră, în care, printre altele, apar şi atributele `CNP` (fireşte, pentru beneficiari – persoane fizice) şi `CodFiscal` (pentru beneficiarii – persoane juridice). Pentru liniile referitoare la persoane fizice `CodFiscal` ar trebui să fie `NULL`, iar pentru persoanele juridice `CNP`-ul ar trebui null-ificat. Însă mulţi practicieni preferă încă valorile zero/spaţiu.

Pentru a verifica dacă atribute/variabile/constante de tip numeric sau şir de caractere sunt necompletate, în sensul că valoarea lor este zero/vidă (nici măcar spaţiu), se poate folosi funcţia `EMPTY()` sau, numai pentru şiruri de caractere, `ISBLANK()`. Pentru o valoare `NULL`, funcţiile `EMPTY()` şi `ISBLANK()` întorc însă `false` (.F.).

Atribuirea de valori vide se poate face, în funcţie de tipul variabilei, astfel:

`X=0` (numeric) sau `x=""` (şir de caractere) sau `x={ / / }` (dată calendaristică).

Dacă un câmp/variabilă primeşte valoarea `NULL`, tipul acestuia/acesteia nu se modifică. Testarea null-ităţii unei variabile/atribut se realizează cu ajutorul comenzii `ISNULL()`.

2.2.4. Crearea şi modificarea structurii unei tabele

Crearea unei tabele reprezintă, de obicei, primul pas în lucrul efectiv cu Visual FoxPro. Graţie unui modul special, proiectantul de tabele (`Table Designer`), lansat prin comanda `CREATE`, sau asistat din meniu, din proiect sau din proiectantul bazei de date (`Database Deeigner`), crearea şi modificarea structurii unei tabele este o operaţiune facilă. Spre exemplu, pentru tabela FACTURI din baza de date VINZARI, proiectantul tabelei se prezintă ca în figura 2.5.

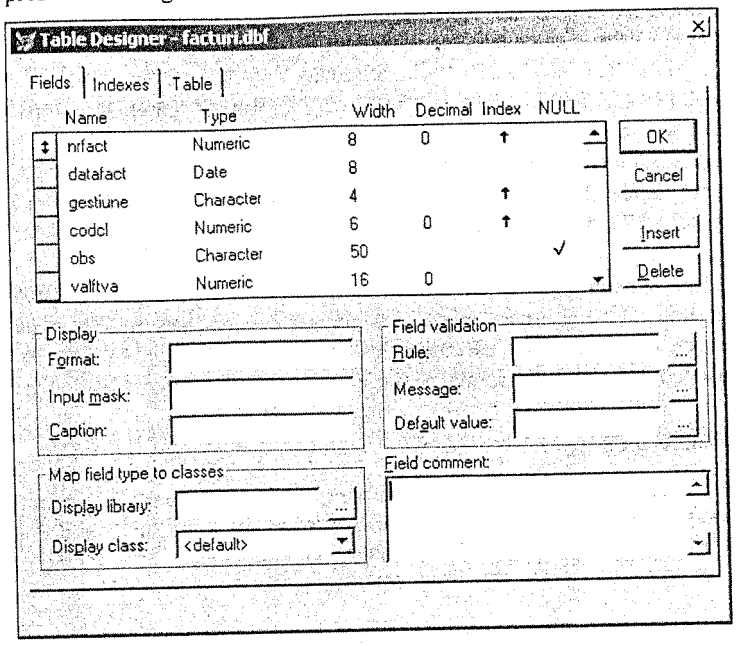


Figura 2.5. Modulul de proiectare de tabele

Definirea mininală a structurii tabelei presupune indicarea numelui fiecărui atribut, a tipului şi lungimii acestuia. Pentru atributele de tip numeric real poate fi precizat şi numărul de poziţii zecimale. Din cele 3 cadre de pagină – `Fields`, `Indexes` şi `Table` – este utilizat mai întâi primul, definirea câmpurilor din tabelă fiind posibilă prin intermediul următoarelor elemente:

`Name` indică numele câmpului. Tabelele independente (neincluse în vreo bază de date) trebuie să respecte limita de 10 caractere pentru numele câmpurilor, specifică versiunilor anterioare VFP 3.0.

- `Type` specifică tipul câmpului.
- `Width` indică numărul maxim de caractere sau de cifre alocat fiecărui câmp.
- `Decimal` precizează numărul de cifre pentru partea fracţionară (când este cazul).
- `Index` specifică indecşii obişnuiţi (`regular`) pentru ordonarea valorilor câmpului ascendent sau descendent.
- `NULL` – când este marcat, atributul respectiv poate accepta valori nule.

Prezenţa dicţionarului (catalogului) bazei de date face posibilă o serie de opţiuni şi „înfloriturí", dintre care merită amintite: folosirea de nume lungi (mai explicite) pentru tabele şi atribute, declararea de valori implicite pentru orice atribut, instituirea unor reguli la nivel de atribut şi înregistrări, ca şi declanşatoare.

Să parcurgem rapid „înfloriturile":

- `Format` – permite definirea şablonului de preluare (tipul de date/caractere) a valorilor atributului respectiv; de obicei se folosesc următoarele caractere: 9 – pentru cifre, A – pentru litere, ! – pentru majuscule etc.
- `Input Mask` – este util în facilitarea preluării valorilor unor câmpuri pentru care se pot defini „măşti"; exemplul din documentaţia VFP este cel al formatului numerelor de telefon din SUA, pentru acest tip de atribute fiind adecvată masca `(###) ###-####`.
- `Caption` – permite declararea unui nume explicit pentru atributul respectiv, nume care să apară la editarea tabelei; de exemplu, pentru `NrFact` să apară în fereastra `Browse`: `NumarFactura`.

Pentru exemplificare, să presupunem că într-o tabelă PERSOANE există un atribut `BI` pentru stocarea seriei şi numărului buletinului de identitate. Cele trei elemente pentru facilitarea preluării/editării valorilor acestui atribut ar putea fi configurate ca în figura 2.6.

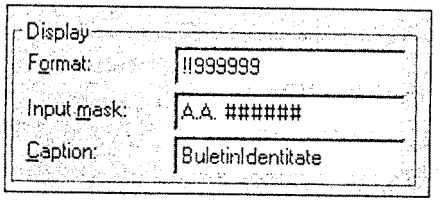


Figura 2.6. Opţiuni de ameliorare a preluării unor date de intrare

`Format`-ul ne asigură că primele două litere preluate (seria buletinului) vor fi majuscule, indiferent de starea tastei `Caps Lock`, iar restul de şase caractere trebuie să fie obligatoriu cifre. `Input Mask` determină afişarea unui punct după fiecare dintre cele două litere şi a unui spaţiu între serie şi număr. `Caption` determină afişarea, în fereastra `Browse`, pentru acest atribut, a antetului `BuletinIdentitate`.

Pentru tabela FACTURI, modul de afişare a antetului şi valorilor atributului `NrFact`, precum şi comentariul referitor la acest câmp sunt cele din figura 2.7.

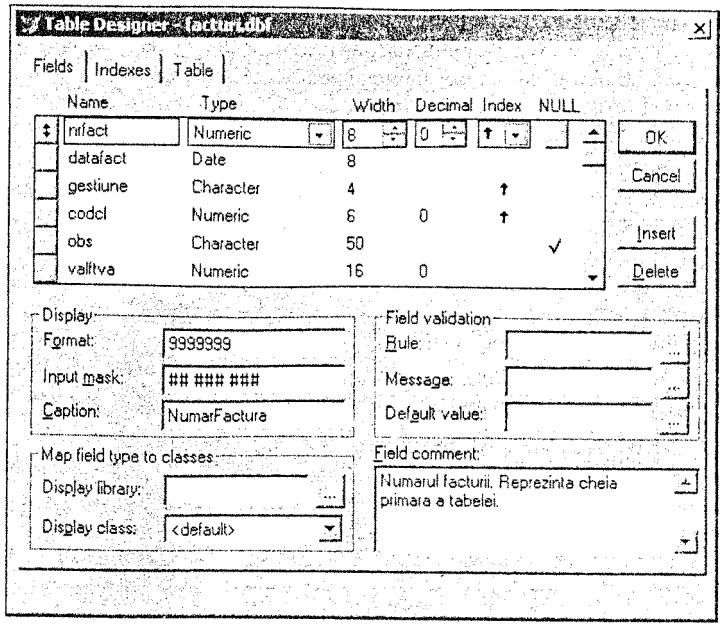


Figura 2.7. Opţiunile de afişare şi de preluare ale atributului `NrFact` din tabela `FACTURI`

După crearea tabelei, aceasta va fi stocată pe disc ca fişier cu extensia `.DBF`. Modificarea ulterioară a structurii, în sensul (restrâns):

- adăugării unui nou atribut,
- ştergerii unui atribut,
- modificării denumirii, tipului sau lungimii unui atribut,
- modificării formatului de afişare etc.

se poate realiza tot cu ajutorul proiectantului de tabelă, lansat fie din meniu sau proiect, fie prin comanda `MODIFY STRUCTURE`.

O tabelă creată prin `Database Designer` este automat inclusă în baza de date curentă, lucru valabil şi dacă baza de date este deschisă prin comanda `OPEN DATABASE`. Asocierea tabelă – bază de date se face în ambele sensuri. În dicţionarul bazei se păstrează calea (directorul) şi numele `.DBF` ce reprezintă, fizic, tabela, iar, pe de altă parte, antetul (*header*-ul) tabelei conţine informaţii despre baza de date de care aparţine (calea şi numele `.DBC`-ului). De aceea, atunci când fişierele dicţionarului de date sau tabele sunt mutate pe altă unitate de disc sau alt director, legăturile stocate în antetul tabelei şi în containerul bazei pot fi alterate.

După o eventuală mutare a unei tabele/baze, actualizarea legăturilor dintre tabelă şi bază se realizează folosind opţiunea `RECOVER` a comenziii `VALIDATE DATABASE`, astfel:

```
OPEN DATABASE vinzari
VALIDATE DATABASE RECOVER
```


Atunci când se doreşte ştergerea marcajului tabelei către baze de date (s-au şters/pierdut fişierele containerului bazei), „eliberarea" tabelei se realizează prin comanda:

```
FREE TABLE tabelă.
```

Firește, după această comandă se pierd restricţiile, declanşatoarele, numele lungi şi alte facilităţi stocate în dicţionar.

Documentaţia VFP recomandă evitarea acestei comenzi atunci când baza există pe disc, deoarece `FREE TABLE` poate face baza inutilizabilă. Din acest punct de vedere, comanda `REMOVE TABLE` este mult mai „blândă", deoarece elimină din bază numai referinţele către indecşii primari, valorile implicite şi regulile de validare asociate tabelei.

2.2.5. Restricţiile unei baze de date

Există două aspecte complementare de abordare a bazelor de date relaţionale: *schema* (structura, intensia) şi *conţinutul* (instanţierea, extensia). Conţinutul unei relaţii este reprezentat de ansamblul tuplurilor ce o alcătuiesc la un moment dat. Pe parcursul exploatării bazei, conţinutul poate creşte liniar (sau exponenţial), în funcţie de volumul şi complexitatea operaţiunilor consemnate. Schema relaţională poate fi definită ca un ansamblu de relaţii asociate *semantic* prin domeniul lor de definiţie şi prin restricţii de integritate. Schema este independentă de timp şi reprezintă componenta permanentă a relaţiilor.

De multe ori se operează numai cu schema simplificată ce cuprinde numele tabelelor şi enumerarea atributelor acestora, atributele-chei primare fiind subliniate. La acestea se adaugă restricţiile la nivel de atribut, înregistrare şi cele referenţiale.

2.2.6. Cheia primară

În teoria bazelor de date relaţionale se interzice apariţia în tabele a două linii (tupluri) absolut identice. Aceasta deoarece accesul într-o tabelă trebuie asigurat exclusiv pe baza valorilor atributelor, iar aceasta presupune ca, pentru delimitarea unei linii de toate celelalte, să existe măcar o valoare care să facă diferenţa.

Atributul sau combinaţia de atribute care nu poate avea valori duplicate în cadrul unei tabele reprezintă un candidat la cheia primară a relaţiei (tabelei). Dintre cheile candidat se alege una care va fi declarată cheie primară, celelalte căpătând statutul de chei alternative. Spre exemplu, într-o tabelă PERSONAL ce conţine câte o linie pentru fiecare angajat al firmei pot exista două atribute candidat: `CNP` – codul numeric personal şi `BI` – seria şi numărul buletinului de identitate. Dacă se alege drept cheie primară `CNP`, atunci `BI` devine cheie alternativă.

În teoria relaţională se *interzice ca un atribut ce este sau intră în componenţa cheii primare să prezinte valori nule.*

2.2.7. Restricţia referenţială

Proiectarea bazei de date se materializează într-o serie de tabele aflate în legătură. De cele mai multe ori, legătura se stabileşte prin intermediul unui atribut cu aceeaşi semnificaţie (şi nume). Să luăm două tabele, JUDETE şi LOCALITATI, cu atributele:

```
JUDETE (Jud, Judet, Regiune)
LOCALITATI (CodPost, Loc, Jud)
```

Cele două tabele prezintă un atribut comun – Jud – care reprezintă indicativul auto al județului. În JUDETE atributul Jud este cheie primară, deoarece identifică fără ambiguitate un județ (cheie alternativă fiind Judet – denumirea județului). Cheia primară a tabelei LOCALITATI este codul poștal – CodPost, iar în aceeași tabelă Jud este cheie străină sau coloană de referință. Tabelele se află într-o relație părinte-copil prin atributul Jud. Tabela în care acest atribut este cheie candidat reprezintă părintele, iar cealaltă, în care Jud este cheie străină, reprezintă copilul.

Dacă în relația copil (LOCALITATI) se interzice apariția de valori nenule ale atributului Jud, care există deja în tabela-părinte (JUDETE), se spune că între cele două tabele s-a instituit o *restricție de integritate referențială* sau, pe scurt, *restricție referențială*.

2.2.8. Alte tipuri de restricții ale bazei de date

Dintre celelalte tipuri de restricții ce pot fi declarate într-o bază de date relațională, merită amintite restricțiile la nivel de atribut și la nivel de înregistrare ce vor fi prezentate în paragraful 2.5.

2.3. Tabelele bazei de date VINZARI

Pe parcursul acestei lucrări, va fi folosită preponderent o bază de date denumită VINZARI, ce conține următoarele tabele:

JUDETE {Jud, Judet, Regiune}

LOCALITATI {CodPost, Loc, Jud}

PERSOANE {CNP, Nume, Prenume, Adresa, Sex, CodPost, TelAcasa, TelBirou, TelMobil, EMail}

CLIENTI {CodCl, DenCl, CodFiscal, Adresa, CodPost, Telefon}

PERSCLIENTI {CNP, CodCl, Functie}

PRODUSE {CodPr, DenPr, UM, Grupa, ProcTVA, Imagine}

GESTIUNI {Gestiune, Den_Gest, Adresa, CodPost, CNP, Telefon, EMail}

FACTURI {NrFact, DataFact, Gestiune, CodCl, Obs}

LINIIFACT {NrFact, Linie, CodPr, Cantitate, PretUnit}

INCASARI {CodInc, DataInc, CodDoc, NrDoc, DataDoc}

INCASFACT {CodInc, NrFact, Transa}

Tabela JUDETE (figura 2.8) conține informații generale despre județele în care sunt clienți. Fiecare linie a tabelei descrie un județ. Atributele sunt:

- Jud – indicativul auto al județului (alcătuit din două litere, majuscule);
- Judet – denumirea județului;
- Regiune – regiunea istorică (provincia) din care face parte județul.

Jud	Judet	Regiune
IS	Iasi	Moldova
VN	Vrancea	Moldova
NT	Neamt	Moldova
SV	Suceava	Moldova
VS	Vaslui	Moldova
TM	Timis	Banat

Figura 2.8. Tabela JUDETE

Cheia primară este atributul Jud. Atributul Judet este cheie alternativă. Pot fi instituite o serie de restricții-utilizator:

- Jud este alcătuit numai din majuscule (eventual un spațiu, pentru București);
- Fiecare cuvânt din Judet începe cu majusculă; restul literelor sunt mici;
- Regiune începe cu majusculă; restul literelor sunt mici;
- Regiune poate avea numai una dintre valorile: Banat, Dobrogea, Muntenia, Oltenia, Transilvania, Moldova.

Tabela LOCALITATI (figura 2.9) conține câte o linie pentru fiecare oraș sau comună. Atenție: satele nu sunt preluate în această tabelă, deoarece, într-o comună, fiecare sat component are același cod poștal ca și comuna !

Atribute:

- CodPost – codul poștal al comunei sau orașului;
- Loc – denumirea comunei/orașului;
- Jud – indicativul auto al județului.

Codpost	Loc	Jud
6600	Iasi	IS
5725	Pascani	IS
6500	Vaslui	VS
5300	Focsani	VN
6400	Birlad	VS
5800	Suceava	SV
5550	Roman	NT
1900	Timisoara	TM

Figura 2.9. Tabela LOCALITATI

Cheia primară este atributul CodPost. Atributul Jud este cheie străină, tabela-părinte fiind JUDETE (prin atributul Jud). Pot fi instituite o serie de restricții-utilizator:

- Jud este alcătuit numai din majuscule (eventual un spațiu, pentru București);
- Fiecare cuvânt din Localitate începe cu majusculă; restul literelor sunt mici.

Tabela CLIENTI (figura 2.10) grupează date generale ale clienților firmei pentru care s-au constituit baze de date (o linie – un client). Atribute:

- `CodCl` – codul clientului;
- `DenCl` – denumirea clientului (persoană juridică);
- `CodFiscal` – codul fiscal;
- `Adresa` – adresa sediului firmei client;
- `CodPost` – codul poştal al comunei sau oraşului;
- `Telefon` – telefonul (principal) al clientului.

CodCl	DenCl	CodFiscal	Adresa	CodPost	Telefon
1001	Client 1 SRL	R1001	Tranzitiei, 13 bis	6600	.NULL.
1002	Client 2 SA	R1002		6600	032-212121
1003	Client 3 SRL	R1003	Prosperitatii, 22	6500	035-222222
1004	Client 4		Sapientei, 56	5725	
1005	Client 5 SRL	R1005	.NULL.	1900	056-111111
1006	Client 6 SA	R1006	Pacientei, 33	5550	.NULL.
1007	Client 7 SRL	R1007	Victoria Capitalismului, 2	1900	056-121212

Figura 2.10. Tabela `CLIENTI`

Cheia primară este atributul `CodCl`. Atributul `CodPost` este cheie străină, tabela-părinte fiind LOCALITATI (prin atributul `CodPost`). `DenCl` şi `Adresa` încep cu majuscule. După cum punctam anterior, valorile nule indică o lipsă de informaţie. Pentru primul client nu se cunoaşte telefonul, celui de-al doilea – adresa etc.

Tabela PERSOANE (figura 2.11) are ca obiectiv stocarea datelor despre persoanele-cheie de la firmele-client: directori generali, directori financiari, şefi ai compartimentelor comerciale (aprovizionare şi/sau vânzări) etc., dar şi ale pesonalului propriu (în primul rând gestionari). Atribute:

- `CNP` – codul numeric al persoanei;
- `Nume`;
- `Prenume`;
- `Adresa`;
- `Sex`;
- `CodPost`;
- `TelAcasă` – numărul telefonului de acasă;
- `TelBirou` – numărul telefonului (fix) de la birou;
- `TelMobil` – numărul „mobilului";
- `Email` – adresa e-mail.

Cnp	Nume	Prenume	Adresa	Sex	Codpost	Telacasa	Telbirou	Telmobil	Email
CNP2	Vasile	Ion	.NULL.	B	6600	234567	876543	094222223	Ion@a.ro
CNP3	Popovici	Ioana	V.Micle, Bl.I, Sc.B,Ap.2	F	5725	345678	.NULL.	094222224	.NULL.
CNP4	Lazar	Catalon	M.Eminescu, 42	B	6500	456789	.NULL.	094222225	.NULL.
CNP5	Iurea	Simion	I.Creanga, 44 bis	B	6500	567890	543210	.NULL.	.NULL.
CNP6	Vasc	Simona	M.Eminescu, 13	F	5725	.NULL.	432109	094222227	.NULL.
CNP7	Popa	Ioanid	I.Ion, Bl.H2, Sc.C, Ap.45	B	1900	789012	321098	.NULL.	.NULL.
CNP8	Bogacs	Ildiko	I.V.Viteazu, 67	F	5550	890123	210987	094222229	.NULL.
CNP9	Ioan	Vasilica	Gara, Bl.B4, Sc.A, Ap.1	F	1900	901234	109876	094222230	.NULL.
CNP10	Tucaliuc	Simion	.NULL.	B	6600	222222	212121	.NULL.	simcrk@mail.dntis.ro
CNP11	Vasiliu	Mihai	Stefan cel Mare, 34	B	5800	258965	256985	0335555555	mihaev@assist.ro
CNP12	Ciubotaru	Toader	Carpati, 12	B	5725	454545	254785	.NULL.	.NULL.

Figura 2.11. Tabela `PERSOANE`

Cheia primară este atributul `CNP`. Fiind vorba şi de persoane exterioare întreprinderii, un atribut precum `Marca` este mai puţin indicat. `CNP`-ul este greoi, relativ lung (de aceea am preferat să scriem, pur şi simplu `CNP1`, `CNP2`...), dar stabil şi unic. Atributul `CodPost` este cheie străină, tabela-părinte fiind `LOCALITATI` (prin atributul `CodPost`). Celei de-a doua persoane nu i se cunoaşte domiciliul (adresa). Cu excepţia lui Vasile Ion, nimeni nu are cont deschis pentru poşta electronică.

Alte restricţii-utilizator:

- Fiecare cuvânt din `Nume` şi `Prenume` începe cu majusculă; restul literelor sunt mici;
- `Adresa` începe cu majusculă;
- Atributul `Sex` poate lua numai două valori: *B* de la *Bărbătesc* şi *F* de la *Femeiesc*.

Tabela PERSCLIENTI (figura 2.12) indică funcţia deţinută de fiecare persoană la unul (sau mai mulţi, deşi aceste cazuri sunt rare) clienţi. O linie din această tabelă reflectă o funcţie deţinută de o persoană la un client. O persoană poate avea mai multe funcţii, chiar la aceeaşi firmă (cumul de funcţii). Cele trei atribute sunt:

- `CNP` – persoana care deţine funcţia;
- `CodCl` – firma la care persoana deţine funcţia;
- `Functie` – funcţia deţinută.

Cnp	Codcl	Functie
CNP1	1001	Director general
CNP2	1002	Director general
CNP3	1002	Sef aprovizionare
CNP4	1003	Sef aprovizionare
CNP5	1003	Director financiar
CNP6	1004	Director general
CNP7	1005	Sef aprovizionare
CNP8	1006	Director financiar
CNP9	1007	Sef aprovizionare

Figura 2.12. Tabela `PERSCLIENTI`

Cheia primară este compusă din toate cele trei atribute: `CNP+CodCl+Functie`, deoarece am convenit că o aceeaşi persoană poate cumula două sau mai multe funcţii la

aceeaşi firmă. CNP şi CodCl sunt chei străine, tabelele-părinte fiind PERSOANE, respectiv CLIENTI. Atributul Functie începe cu majusculă, restul literelor fiind mici.

Tabela PRODUSE (figura 2.13) reprezintă nomenclatorul de produse şi servicii pe care le comercializează firma (produsele sunt obţinute prin manufacturare sau revânzare). Atribute:

- CodPr – codul produsului;
- DenPr – denumirea;
- UM – unitatea de măsură a produsului;
- Grupa – grupa de mărfuri (sortimente) în care se încadrează; acest atribut este important pentru analiza vânzărilor;
- ProcTVA – procentul TVA care se aplică la preţul de vânzare (preţ care este fără TVA); pare de prisos în condiţiile actuale, când toate produsele au un singur procent, 19%, dar nimeni nu poate garanta cum vor gândi promoţiile viitoare de guvernanţi „relaxarea fiscală";
- Imagine – este un câmp de tip general folosit la stocarea fotografiei digitale a fiecăruia dintre produsele vândute.

Codpr	Denpr	Um	Grupa	Proctva	Imagine
1	Produs 1	buc	Tigari	.19	gen
2	Produs 2	kg	Bere	.19	gen
3	Produs 3	kg	Bere	.19	gen
4	Produs 4	l	Dulciuri	.19	gen
5	Produs 5	buc	Tigari	.19	gen

Figura 2.13. Tabela PRODUSE

Cheia primară este atributul CodPr. Alte restricţii-utilizator:

- DenPr şi Grupa încep cu majusculă; restul literelor sunt mici;
- unitatea de măsură (UM) se scrie numai cu litere mici.

Tabela GESTIUNI (figura 2.14) conţine câte o linie pentru fiecare gestiune de produse/mărfuri a firmei, gestiune din care se fac vânzări (un soi de puncte de vânzare ale firmei). Atribute:

- Gestiune – codul gestiunii/depozitului;
- Den_Gest – denumirea gestiunii;
- Adresa – adresa depozitului;
- CodPost – codul poştal al localităţii în care se află depozitul;
- CNP – codul numeric personal al gestionarului;
- Telefon – numărul de telefon al depozitului;
- Email – adresa de e-mail a depozitului.

Gestiune	Den_gest	Adresa	Codpost	Cnp	Telefon	Email
001	Depozit Pacurari	Sos. Pacurari, nr 145 bis	6600	CNP10	212121	x.gest001@mail.dntis.ro
002	Depozit Suceava	Str. Lunetistului nr.33	5800	CNP11	256985	x.gest002@mail.dntis.ro
003	Sectie Pascani	Bd. Trompetistului, 56	5725	CNP12	254785	x.gest003@mail.dntis.ro

Figura 2.14. Tabela GESTIUNI

Cheia primară a tabelei este atributul Gestiune.

Tabela FACTURI (figura 2.15) conţine câte o linie pentru fiecare factură emisă, factură ce reflectă o vânzare (către un client). Atribute:

- NrFact – numărul facturii;
- DataFact – data întocmirii facturii;
- Gestiune – codul gestiunii/depozitului din care se face vânzarea;
- CodCl – codul clientului căruia i s-au vândut produsele/serviciile consemnate în factură;
- Obs – observaţii; e folosit relativ rar, pentru a introduce eventuale detalii sau probleme care au apărut în legătură cu o factură.

NumarFactura	Datafact	Gestiune	Codcl	Obs
1 111	01/08/2001	001	1001	
1 112	01/08/2001	001	1005	Probleme cu transportul
1 113	01/08/2001	001	1002	
1 114	01/08/2001	002	1006	
1 115	02/08/2001	002	1001	
1 116	02/08/2001	001	1007	Pretul propus initial a fost modificat
1 117	03/08/2001	002	1001	
1 118	04/08/2001	003	1001	
1 119	07/08/2001	003	1003	
1 120	07/08/2001	003	1001	
1 121	07/08/2001	001	1004	
1 122	07/08/2001	003	1005	

Figura 2.15. Tabela FACTURI

Cheia primară este atributul NrFact. Explicaţia este una simplă: gestionarea facturilor emise este, conform legii, strictă. Nu pot exista două facturi (care reflectă două vânzări) cu acelaşi număr. Sau, de fapt, pot exista, dar aceasta înseamnă o ilegalitate destul de gravă (se mai poartă încă prin economia subterană). CodCl este cheie străină (tabela-părinte este CLIENTI). Ca restricţie-utilizator suplimentară, nu pot fi introduse facturi întocmite înainte de 1 august 2001 (DataFact >= {01/08/2001}).

Tabela LINIIFACT (figura 2.16) detaliază tabela precedentă. Un tuplu se referă la un produs/serviciu vândut şi consemnat într-o factură emisă. Pentru fiecare factură vor fi atâtea linii câte produse/servicii au fost consemnate la vânzarea respectivă. Atribute:

- NrFact – numărul facturii;
- Linie – numărul liniei din factura respectivă;

- CodPr – codul produsului/serviciului vândut;
- Cantitate – cantitatea vândută;
- PretUnit – prețul unitar (fără TVA) la care s-a făcut vânzarea.

Nrfact	Linie	Codpr	Cantitate	Pretunit
1111	1	1	50	10000
1111	2	2	75	10500
1111	3	5	500	6500
1112	1	2	80	10300
1112	2	3	40	7500
1113	1	2	100	9750
1114	1	2	70	10700
1114	2	4	30	15800
1114	3	5	700	6400
1115	1	2	150	9250
1116	1	2	125	9300
1117	1	2	100	10000
1117	2	1	100	9500
1118	1	2	30	11000
1118	2	1	150	9300
1119	1	2	35	10900
1119	2	3	40	7000
1119	3	4	50	14000
1119	4	5	750	6300
1120	1	2	80	11200
1121	1	5	550	6400
1121	2	1	100	10500

Figura 2.16. Tabela LINIIFACT

Cheia primară este combinația NrFact+Linie. Atributele NrFact și CodPr sunt chei străine. Pentru a determina valoarea de încasat a unei facturi (inclusiv TVA), la valoarea fără TVA, pentru fiecare linie (obținută prin produsul Cantitate * PretUnit) trebuie adăugată TVA colectată, obținută prin aplicarea procentului de TVA al produsului/serviciului (atributul ProcTVA din PRODUSE) la valoarea fără TVA.

Tabela INCASARI (figura 2.17) reprezintă un nomenclator al încasărilor. Printr-o încasare, un client își stinge una sau mai multe obligații de plată, adică achită una sau mai multe facturi. Documentul primar pe baza căruia se consemnează încasarea poate fi ordinul de plată, cecul, chitanța etc. Atributele tabelei sunt:

- CodInc – codul încasării este un număr intern, util pentru a diferenția o încasare de celelalte;
- DataInc – data încasării – data la care banii au intrat în contul sau casieria firmei;
- CodDoc – codul documentului justificativ al încasării: OP – ordin de plată, CHIT – chitanță, CEC – filă cec;
- NrDoc – numărul documentului justificativ;
- DataDoc – data la care a fost întocmit documentul justificativ; din momentul întocmirii documentului de plată până la data la care banii ajung efectiv în

cont/casierie trec câteva zile sau săptămâni (datorită circuitului documentelor între firme și bănci).

Codinc	Datainc	Coddoc	Nrdoc	Datadoc
1234	15/08/2001	OP	111	10/08/2001
1235	15/08/2001	CHIT	222	15/08/2001
1236	16/08/2001	OP	333	09/08/2001
1237	17/08/2001	CEC	444	10/08/2001
1238	17/08/2001	OP	555	10/08/2001
1239	18/08/2001	OP	666	11/08/2001

Figura 2.17. Tabela INCASARI

Cheia primară este atributul CodInc. Ca restricții-utilizator pot fi instituite:

- data încasării nu o poate precede pe cea a întocmirii documentului (DataDoc <= DataInc);
- data documentului de încasare nu poate fi anterioară datei de început a aplicației: 1 august 2000 (DataDoc >= {01-08-2000});
- codul documentului se scrie numai cu majuscule.

Tabela INCASFACT (figura 2.18) detaliază tabela precedentă și indică ce facturi (tranșe din facturi) sunt achitate prin fiecare încasare. Un client poate plăti mai multe facturi o dată (printr-o singură plată). Pe de altă parte, orice factură poate fi plătită în una sau mai multe tranșe, în funcție de banii de care dispune clientul la momentul întocmirii documentului de plată. Atributele tabelei sunt:

- CodInc – codul încasării;
- NrFact – factura pentru care se încasează valoarea integrală sau numai o tranșă;
- Transa – tranșa din factură (sau întreaga valoare) care se încasează prin documentul primar ce stă la baza încasării.

Codinc	Nrfact	Transa
1234	1111	5399625
1234	1118	1026375
1235	1112	487705
1236	1117	975410
1236	1118	1026375
1236	1120	731557
1237	1117	975410
1238	1113	1160250
1239	1117	369680

Figura 2.18. Tabela INCASFACT

Cheia primară este combinația CodInc+NrFact, deoarece la o încasare se pot achita mai multe facturi, iar pe de altă parte, o factură poate fi plătită în mai multe tranșe. CodInc și NrFact sunt chei străine.

2.4. Despre indecşi

Orice produs software dedicat gestionării bazelor de date este de neconceput fără opţiuni privind suportul indecşilor. Aceasta deoarece un parametru vital pentru buna funcţionare a unei aplicaţii este viteza.

Tipologia indecşilor este destul de largă, chiar şi în VFP. Noi însă ne vom ocupa doar de cei *compuşi structurali*, care sunt organizaţi în fişiere cu acelaşi nume ca al tabelei, dar cu extensia `.CDX`. Un index compus este alcătuit din unul sau mai mulţi indecşi elementari (`tag`-uri) ce conţin *valorile ordonate* ale câmpului sau combinaţiei de câmpuri ce constituie expresia de indexare, precum *şi adresele înregistrărilor* corespunzătoare din tabela originară.

Indecşii sunt absolut necesari pentru declararea cheilor primare, ale celor alternative şi a restricţiilor referenţiale. Pentru a evidenţia aportul la creşterea vitezei, vom apela la un exemplu edificator (sau cel puţin aşa ni se pare nouă): dată fiind o tabelă `TEST (cod N(10), den_test C(20))` aflată pe un server de aplicaţii în reţea (2×400 Mhz cu 128 RAM, reţea pe 10 Mbps) şi care conţine 2.500.000 de înregistrări, se doreşte obţinerea, pe o staţie de lucru, a acelor înregistrări care au codul cuprins între 999.999 şi 1.239.000. Rezultatul se prezintă astfel:

- pe tabela neindexată interogarea respectivă are o durată de 1 min 10 s,

în timp ce

- pe tabela indexată după `Cod` durata obţinerii rezultatului se reduce la 12 s.

În funcţie de expresia de indexare, indecşii pot fi:

- *simpli* – expresia de indexare este chiar atributul respectiv;
- *complecşi* – expresia de indexare reprezintă o combinaţie de atribute.

Astfel, tabela LINIIFACT `(NrFact N(6), Linie N(2), Codpr N(6), Cantitate N(5), Pretunit N(12) )` (figura 2.19) are drept cheie primară combinaţia între numărul de factură şi cel al liniei. Declararea cheii primare se realizează construind un index cu numele `Primar` şi expresia de indexare `str(nrfact)+str(linie)`. Pentru cheile străine `codpr` (cu referinţă la tabela PRODUSE) şi `nrfact` (cu referinţă la tabela FACTURI) se creează câte un index obişnuit (regular) astfel: `NrFact` cu expresia de indexare `nrfact` şi `CodPr` cu expresia de indexare `codpr`. Prin urmare, fişierul index compus nou – `LINIIFACT.CDX` – va conţine trei indecşi elementari (tag-uri). Fiecare dintre aceşti indecşi va conţine expresiile de indexare ordonate şi adresele înregistrărilor din tabelă (figura 2.20).

Liniifact

Adresa_inreg	Nrfact	Linie	Codpr	Cantitate	Pretunit
1	1111	1	1	50	10000
2	1111	2	2	75	10500
3	1111	3	5	500	6500
4	1112	1	2	80	10300
5	1112	2	3	40	7500
6	1113	1	2	100	9750
7	1114	1	2	70	10700
8	1114	2	4	30	15800
9	1114	3	5	700	6400
10	1115	1	2	150	9250
11	1116	1	2	125	9300
12	1117	1	2	100	10000
13	1117	2	1	100	9500
14	1118	1	2	30	11000
15	1118	2	1	150	9300
16	1120	1	2	80	11200
17	1121	1	5	550	6400
18	1121	2	1	100	10500
19	1121	4	5	4	5
20	1119	1	2	35	10900
21	1119	2	3	40	7000
22	1119	3	4	50	14000
23	1119	4	5	750	6300

Figura 2.19. Tabela `LINIIFACT` cu tot numărul înregistrării

Tag_primar

Adresa_inreg	Primar
1	1111 1
2	1111 2
3	1111 3
4	1112 1
5	1112 2
6	1113 1
7	1114 1
8	1114 2
9	1114 3
10	1115 1
11	1116 1
12	1117 1
13	1117 2
14	1118 1
15	1118 2
20	1119 1
21	1119 2
22	1119 3
23	1119 4
16	1120 1
17	1121 1
18	1121 2
19	1121 4

Tag_nrfact

Adresa_inreg	Nrfact
1	1111
2	1111
3	1111
4	1112
5	1112
6	1113
7	1114
8	1114
9	1114
10	1115
11	1116
12	1117
13	1117
14	1118
15	1118
20	1119
21	1119
22	1119
23	1119
16	1120
17	1121
18	1121
19	1121

Tag_codpr

Adresa_inreg	Codpr
1	1
13	1
15	1
18	1
2	2
4	2
6	2
7	2
10	2
11	2
12	2
14	2
16	2
20	2
5	3
21	3
8	4
22	4
3	5
9	5
17	5
19	5
23	5

Figura 2.20. Organizarea internă a fişierului index `LINIIFACT.CDX`

În Visual FoxPro 6.0 pot fi utilizate următoarele **tipuri de indecşi**:

- **Primar** – pentru declararea cheii primare – blochează introducerea de valori duble pentru expresia de indexare.
- **Candidat** – acelaşi efect ca şi indexul primar. Deşi pare oarecum redundant, este nevoie şi de acest tip, deoarece o tabelă poate avea mai multe chei candidat la postul de identificator primar (ex.: marca salariat şi cod numeric personal).
- **Regular** – permite duplicarea valorilor, fiind folosit pentru cheile străine, precum şi pentru construirea unor indecşi necesari creşterii vitezei accesului la înregistrări.
- **Unique** – inclus pentru compatibilitatea cu versiunile mai vechi de FoxPro, acesta generează şi ordonează în fişierul index doar valorile unice ale expresiei de indexare, în cazul în care respectivele valori se dublează în tabela de bază.

2.4.1. Crearea indecşilor în mod vizual

Pentru crearea asistată a indecşilor vom parcurge următorii paşi:

1. în `Project Manager` se selectează cadrul de pagină `Data`, apoi baza de date şi tabela pentru care doriţi să creaţi un index şi se execută un click pe butonul `Modify`;
2. în `Table Designer` se alege cadrul de pagină `Indexes`.

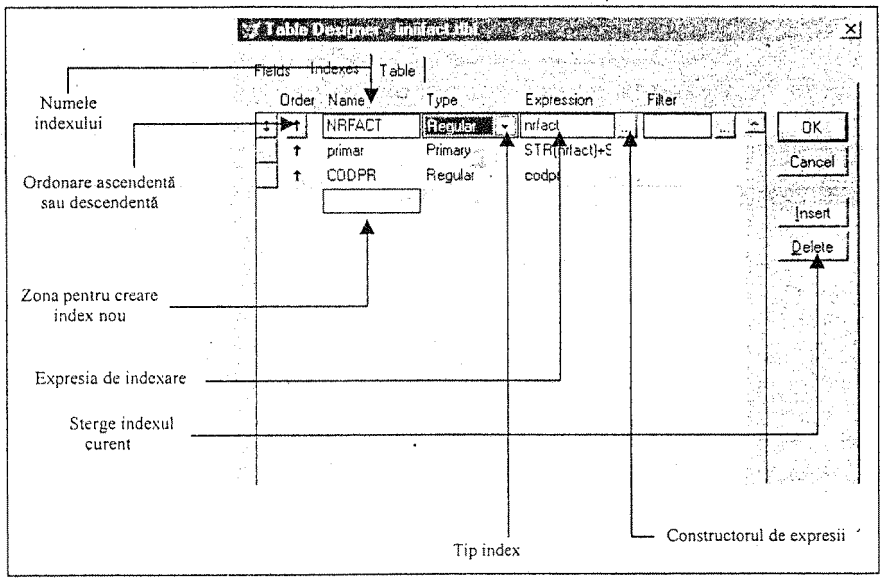


Figura 2.21. Cadrul de pagină pentru construirea asistată a indecşilor

După cum se observă şi din figura 2.21, nu este obligatoriu ca numele indexului să fie diferit de expresia de indexare (nume index – NRFACT, expresia de indexare – `nrfact`). Pentru indexul complex cu numele `Primar`, expresia de indexare se constituie din concatenarea câmpurilor `nrfact` şi `linie` transformate în caracter cu ajutorul funcţiei `STR()`: `str(nrfact)+str(linie)`.

2.4.2. Crearea indecşilor prin comenzi

În funcţie de tipul indexului dorit, avem următoarele situaţii:

- Indecşi de tip *regular*, *candidate* sau *unique*:

```
INDEX ON expresie_de_indexare TAG numeindex
      [ASCENDING / DESCENDING]
      [UNIQUE / CANDIDATE]
```

Utilizarea comenzii de mai sus necesită deschiderea şi selectarea zonei de lucru în care este deschisă tabela curentă (vezi capitolul 3).

`<expresie_de_indexare>` – poate fi un atribut sau o combinaţie de atribute (pentru indecşi complecşi)

- Indecşi primari (cheile primare) şi indecşi pentru cheile străine:

```
ALTER TABLE numetabelă  ADD PRIMARY KEY
      expresie_de_indexare TAG Nume_Index

ALTER TABLE numetabelă  ADD FOREIGN  KEY
      expresie_de_indexare TAG Nume_Index  REFERENCES
      NumeTabela_Parinte TAG Nume_Index_Parinte
```

Comanda `ALTER TABLE`, fiind o comandă SQL (tratată pe larg în capitolul 5), va deschide automat tabela, astfel încât nu mai este necesară efectuarea acestei operaţii în mod explicit.

Atenţie! Adăugarea cheilor străine prin această metodă realizează *numai legăturile permanente între tabelele părinte şi copil*, modul de comportament specific modelului relaţional fiind stabilit ulterior prin *editarea restricţiilor referenţiale* sau a *declanşatoarelor* (triggere)[13].

Pentru exemplificare, vom crea indecşii tabelei LINIIFACT:

- crearea cheilor primare şi străine:

```
ALTER TABLE liniifact ADD PRIMARY KEY str(nrfact)+str(linie) TAG primaru
ALTER TABLE liniifact ADD FOREIGN KEY nrfact TAG nrfact REFERENCES Facturi TAG nrfact
ALTER TABLE liniifact ADD FOREIGN KEY codpr TAG codpr REFERENCES Produse; TAG codpr
```

- crearea unui index simplu pentru atributul `Linie`, spre exemplu, presupune şi deschiderea[14] tabelei (dacă aceasta nu este deja deschisă):

```
SELECT liniifact
INDEX ON linie TAG linie
```

Rezultatul acestor linii de cod se concretizează în următoarele:

- Se creează indexul primar `Primaru` cu expresia de indexare `str(nrfact)+str(linie)`.

13. Vezi paragraful 2.5 al acestui capitol.
14. Vezi capitolul 3.

- Se creează indecşii obinşuiţi („regulari") pentru cheile străine cu numele `Nrfact`, `Codpr` şi expresiile de indexare `nrfact`, `codpr`, indecşi care leagă tabela LINIIFACT cu tabelele-părinte PRODUSE şi FACTURI prin indecşii corespunzători. Este de la sine înţeles (sperăm) că, înaintea construirii cheilor străine pentru tabela LINIIFACT, este obligatoriu să construim indecşii de legătură pe tabelele-părinte PRODUSE şi FACTURI.
- Se creează indexul obişnuit (regular) cu numele `Linie` şi expresia de indexare atributul cu acelaşi nume – `linie` –, cu scopul de a spori viteza de procesare a înregistrărilor.
- Toţi indecşii creaţi se regăsesc în fişierul LINIIFACT.CDX.

Cum construim indecşii complecşi ?

Pentru construirea indecşilor complecşi, de multe ori este necesară convertirea valorilor atributelor în şiruri de caractere. Pentru cheile primare, această operaţiune este obligatorie. Avem aşadar următoarele situaţii:

- Toate atributele care constituie indexul sunt de tip caracter – rezultă expresia de indexare : `atribut1+atribut2+ ...`
- `Atribut1` este de tip numeric, `atribut2` de tip caracter, iar `atribut3` de tip dată calendaristică etc... – rezultă expresia de indexare:
  `STR(atribut1)+atribut2+DTOS(atribut3).`

2.4.3. Probleme şi erori legate de indecşi

Aportul indecşilor la buna funcţionare a unei aplicaţii este esenţial. Acesta este şi motivul pentru care deteriorarea lor constituie o preocupare majoră a celor care răspund de buna funcţionare a software-ului dezvoltat în VFP. Din păcate, indecşii constiuie un adevărat călcâi („crăpat") al lui Ahile. Motorul de bază de date Visual FoxPro nu actualizează întotdeauna la timp indecşii în pas cu actualizările efectuate în tabela de bază. Cu alte cuvinte, uneori indecşii „rămân în urmă" faţă de situaţia existentă la un moment dat în tabelă.

Cea mai frecventă eroare legată de acest aspect, pe care o vom întâlni în practica de toate zilele, o regăsim în figura 2.22.

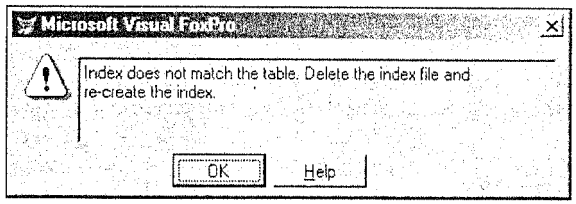


Figura 2.22. Mesaj de eroare pentru un index „crăpat"

Eroarea este cu atât mai apăsătoare cu cât de multe ori apare „din senin" şi nu permite nici măcar deschiderea tabelei în cauză. Negăsind, până acum, nici o explicaţie privind cauzele producerii acestei erori (oricum o regăsim mai des în aplicaţiile de mai mare amploare, cu un volum ridicat de tranzacţii pe o perioadă scurtă de timp), vom prezenta două modalităţi de tratare a acesteia.

Întrucât tabela GESTIUNI face parte din baza de date VINZARI, se poate recurge la următorii paşi:

- în fereastra de comandă introducem secvenţa:

```
OPEN DATABASE vinzari
VALIDATE DATABASE RECOVER
```

Această comandă verifică integritatea şi existenţa fizică pe disc (acolo unde este cazul) a tuturor obiectelor bazei de date. În cazul în care un obiect este invalid şi nu poate fi refăcut automat, există posibilitatea ştergerii acestuia din containerul bazei de date şi reconstruirii lui. În cazul nostru va apărea mesajul de eroare din figura 2.23:

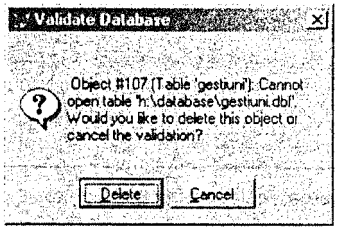


Figura 2.23. Mesaj de eroare la validarea BD în cazul inconsistenţei unui index

- se apasă butonul `Delete`, care va şterge obiectul de tip tabelă `GESTIUNI` din containerul bazei de date (tabela rămâne pe disc, dar nu se mai află în baza de date);
- se şterge de pe disc (fizic) fişierul `GESTIUNI.CDX` (fişierul index care nu mai corespunde cu tabela). Atenţie, nu trebuie ştearsă tabela `GESTIUNI.DBF`;
- în `Project Manager`, ne poziţionăm în zona tabelelor şi apăsăm butonul ADD, buton care adaugă în containerul bazei de date o tabelă aflată pe disc dar care nu este deja în baza de date (aşa-numitele tablele libere). În formularul de căutare deschis se alege `GESTIUNI.DBF` (figura 2.24);

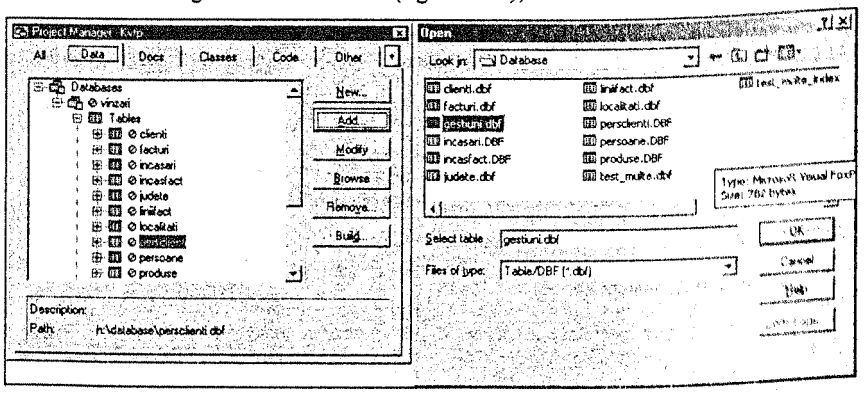


Figura 2.24. Adăugarea unei tabele „libere" existente pe disc la containerul bazei de date

- se reconstruiesc indecşii folosind una dintre modalităţile prezentate anterior şi se refac restricţiile referenţiale sau se creează/adaugă declanşatoarele (**triggerele**) tabelei (figura 2.25).

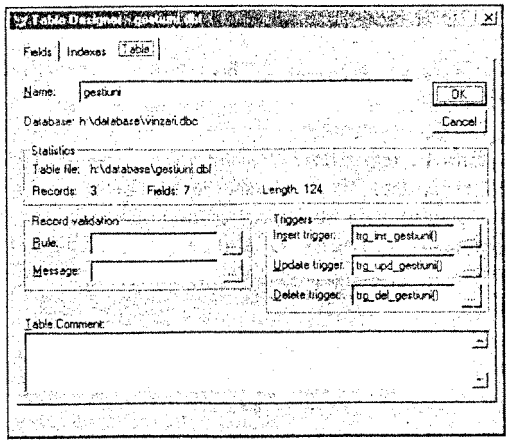


Figura 2.25. Adăugarea declanşatoarelor tabelei `GESTIUNI` în mod asistat

A doua modalitate de remediere a situaţiei este utilizarea unei secvenţe de comenzi specifice. Această metodă este mai mult decât recomandată în cazul unor aplicaţii finalizate şi furnizate clienţilor, deoarece nu necesită existenţa unei versiuni instalate de Visual FoxPro şi poate fi introdusă sub forma unei opţiuni în meniul aplicaţiei (sau într-un formular), pentru a fi accesată rapid de utilizator în caz de nevoie. Pentru cazul tabelei GESTIUNI, procedura o regăsim în listingul 2.1.

Listing 2.1. Refacerea indecşilor tabelei `GESTIUNI` prin program

```
OPEN DATABASE vinzari
REMOVE TABLE gestiuni
ERASE DATABASE/gestiuni.CDX
ADD TABLE gestiuni
ALTER TABLE gestiuni ADD PRIMARY KEY gestiune TAG gestiune
ALTER TABLE gestiuni ADD FOREIGN  KEY codpost TAG codpost REFERENCES localitati TAG
codpost
ALTER TABLE gestiuni ADD FOREIGN  KEY cnp TAG cnp REFERENCES persoane TAG cnp

*!* numai pentru cazul în care avem nume de atribute mai lungi de 10 caractere,
*!* prin scoaterea tabelei din baza de date tot ce depaseste 10 caractere
*!* se pierde fiind necesara si o comanda de redenumire a atributelor:

ALTER TABLE gestiuni RENAME COLUMN telefon_pt TO telefon_pttr

*!* adaugarea declansatoarelor :
CREATE TRIGGER ON gestiuni FOR UPDATE AS trg_upd_gestiuni()
CREATE TRIGGER ON gestiuni FOR DELETE AS trg_del_gestiuni()
CREATE TRIGGER ON gestiuni FOR INSERT AS trg_ins_gestiuni()
```


2.5. Restricţiile declarate într-o bază de date VFP

Pentru asigurarea integrităţii şi coerenţei bazei de date, precum şi pentru a facilita preluarea datelor, ceea ce înseamnă şi un spor de productivitate, în Visual FoxPro se pot declara diferite tipuri de restricţii, ce decurg fie din regulile specifice modelului relaţional, fie în urma aplicării anumitor reguli ce ţin de domeniul de implementare al aplicaţiei. Dacă în versiunile mai vechi ale FoxPro toate aceste reguli erau implementate exclusiv prin intermediul interfeţei-utilizator, în Visual FoxPro le putem introduce direct în definiţia tabelei. Ca urmare, restricţiile vor fi memorate în dicţionarul bazei de date şi va fi imposibilă încălcarea lor chiar prin actualizări ce ocolesc interfaţa şi se execută direct pe o tabelă.

Tipurile de reguli ce pot fi definite astfel sunt:

- *restricţii la nivel de atribut şi/sau înregistrare* ce decurg din reguli specifice mediului de implementare a aplicaţiei;
- *restricţii referenţiale* ce decurg din modelul relaţional.

2.5.1. Definirea restricţiilor la nivel de atribut şi înregistrare

O regulă de validare la nivel de atribut sau înregistrare este o funcţie care „întoarce" o valoare logică, `.T. (TRUE)` sau `.F. (FALSE)`. Dacă valoarea returnată este `TRUE`, atunci modificarea este operată în tabelă. Dacă este `FALSE`, se afişează un mesaj de avertizare, iar operaţiunea prin care s-a încălcat restricţia este anulată.

Reguli la nivel de atribut

Modulul de editare a structurii tabelei (`Table Designer`), pe lângă cele şase elemente care permit definirea atributelor acesteia, prezintă în jumătatea de jos a ferestrei alte două grupuri de opţiuni, `Display` şi `Field validation`. `Display` permite formatarea valorilor atributului la culegere şi afişarea pe ecran. Pentru acest punct al discuţiei interesează cel de-al doilea grup. Opţinea `Rule` permite definirea unei reguli de validare la nivel de atribut, în timp ce `Message` va indica ce mesaj apare pe ecran atunci când regula nu este respectată. Cea de-a treia opţiune, `Default value`, este utilă pentru specificarea valorii implicite a atributului, valoare care va „umple" câmpul respectiv la adăugarea unei noi înregistrări.

Pentru tabela LINIIFACT se stabileşte o regulă de validare la nivelul atributului `Linie`, regulă ce nu va permite introducerea unor valori negative pe câmpul respectiv (figura 2.26).

Atenţie! Pentru că fiecare atribut al tabelei poate avea propria regulă de validare, primul pas va fi poziţionarea pe câmpul căruia dorim să-i definim regula.

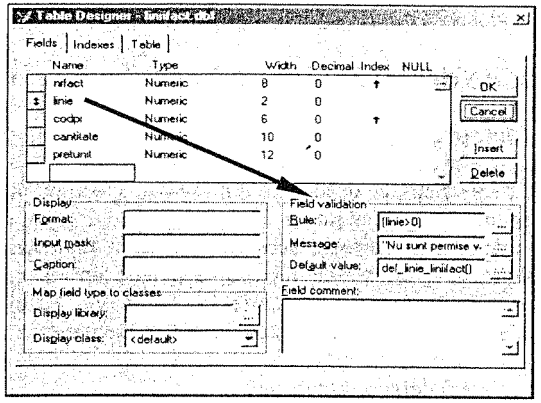


Figura 2.26. Stabilirea regulilor de validare la nivel de atribut

Aşadar, la fiecare actualizare a atributului `Linie` se verifică dacă noua valoare este pozitivă, iar dacă nu, se afişează mesajul de avertizare *Nu sunt permise valori negative pe acest câmp!* şi se restabileşte automat situaţia iniţială. Mesajul trebuie încadrat între ghilimele.

În căsuţa de tip text `Default Value` din imaginea de mai sus avem o funcţie-utilizator (`def_linie_liniifact()`) stocată în dicţionarul bazei de date (procedură stocată[15]) care incrementează automat cu câte o unitate valoarea atributului `Linie` pentru fiecare linie nouă a aceluiaşi număr de factură.

Notă. Valoarea implicită poate fi însă orice valoare care corespunde tipului atributului şi care nu încalcă regulile de validare. Dacă atributul este de tip caracter, şirul de caractere ce va constitui valoarea implicită (dacă nu folosim o funcţie) trebuie încadrat între ghilimele.

Reguli la nivel de înregistrare

Deseori, interesează nu numai valorile pe care le poate avea un atribut, dar şi restricţiile ce privesc relaţiile dintre două sau mai multe câmpuri ale aceleiaşi înregistrări. Revenind la tabela LINIIFACT, instituim următoarea regulă impusă de conducerea firmei: pentru produsul ce are codul 4, cantitatea ce poate apărea pe o factură nu trebuie să depăşească 300 de unităţi. Specific acestei reguli este faptul că implică două câmpuri: atunci când `cod_prod = 4`, `cantitate` e musai `<= 300`. Cum pentru cuvântul-cheie „musai" nu există o clauză specială, în Visual FoxPro regula se scrie cu funcţia `IIF(codpr=4 .AND. cantitate>300, .F., .T.)`. Această expresie se introduce în cadrul de pagină `Table` al proiectantului de tabele (`Table Designer`), după cum se observă în figura 2.27.

15. Vezi capitolul 6, paragraful 6.3.

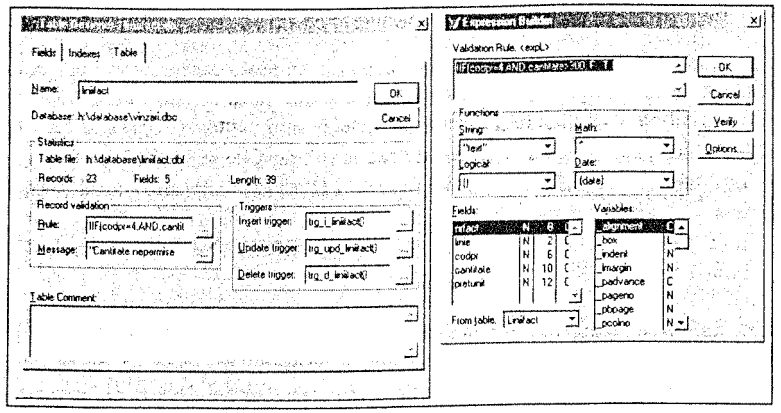


Figura 2.27. Reguli (restricţii) la nivel de înregistrare

Cadrul de pagină `Table` oferă informaţii generoase despre tabelă: numele bazei de date din care face parte, câte înregistrări conţine, câte atribute alcătuiesc tabela şi care este suma lungimilor acestora. Pentru declarea restricţiei la nivel de înregistrare, interesează grupul `Record validation`. În căsuţa de text `Rule` a fost introdusă funcţia descrisă anterior, în timp ce pentru `Message` s-a specificat şirul de caractere: *„Cantitate nepermisa pentru acest produs !!!"*.

Figura 2.28 prezintă modul de comportament al tabelei la încălcarea restricţiilor la nivel atribut şi la nivel înregistrare).

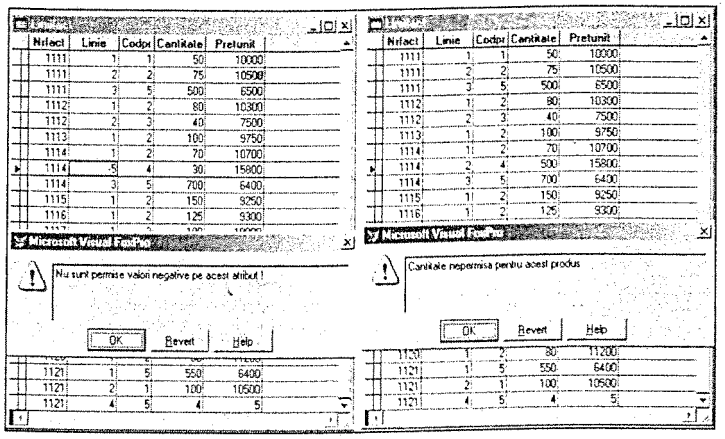


Figura 2.28. Mesajele de avertizare generate la încălcarea restricţiilor la nivel de atribut (stânga) şi la nivel de înregistrare (dreapta)

Atenţie! În momentul introducerii regulilor de validare, toate înregistrările introduse deja în tabelă (înainte de stabilirea regulilor) sunt verificate. Dacă măcar una încalcă restricţiile, regulile nu vor fi salvate în dicţionarul de date. Din acest motiv este bine ca acestea să fie definite înainte de a introduce înregistrări în tabele.

2.5.2. Legăturile permanente dintre tabele şi editarea restricţiilor referenţiale

După cum am văzut anterior (la începutul capitolului), construirea restricţiilor referenţiale în Visual FoxPro se bazează pe indecşi. Din baza de date VINZARI luăm spre exemplificare tabela LOCALITATI, care este tabelă-copil pentru JUDETE şi părinte pentru PERSOANE, GESTIUNI şi CLIENTI. Restricţiile referenţiale se instituie astfel:

- între tabelele JUDETE (părinte) şi LOCALITATI (copil) atributul de legătură este `jud`;
- între tabelele LOCALITATI (părinte) şi PERSOANE, GESTIUNI, CLIENTI (copii) atributul de legătură este `codpost`.

Pentru stabilirea legăturilor permanente (suportul restricţiilor referenţiale) este necesară, pe de o parte, crearea indecşilor primari în tabelele-părinte şi, pe de altă parte, crearea, în fiecare tabelă-copil, a câte unui index de tip `Regular` pentru fiecare dintre atributele de legătură ce constituie chei străine. Pentru simplificare, numele indecşilor „obişnuiţi" au acelaşi nume ca şi cel al atributului din expresie.

Cel mai simplu mod de creare a legăturilor permanente între tabele se realizează din proiectantul bazei (`Database Designer`) printr-o simplă „tragere" de mouse *pornind de la numele indexului primar* din tabela-*părinte spre numele indexului* „obişnuit" (*regular*) corespondent al tabelei-*copil*. Figura 2.29 ilustrează acest lucru.

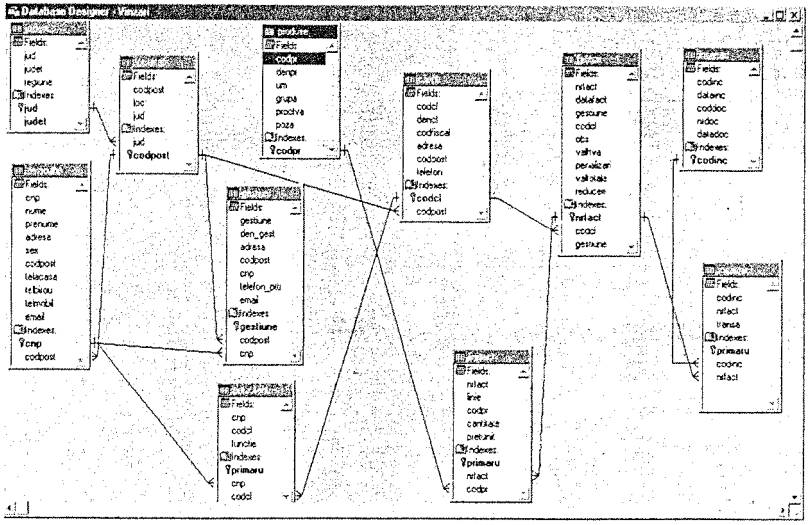


Figura 2.29. Crearea legăturilor permanente între tabele în mod asistat

Linia ce uneşte tabelele JUDETE şi LOCALITATI este simplă „înspre" părinte şi „trifurcată" „înspre" copil (LOCALITATI). Această înseamnă că relaţia dintre cele două tabele este de tip *one-to-many* (una-la-mai multe).

Atenţie! Declararea legăturii permanente între tabele nu echivalează cu instituirea restricţiilor referenţiale! În VFP, stabilirea restricţiilor referenţiale presupune indicarea modului în care SGBD-ul va „reacţiona" la actualizarea tabelei-părinte şi a celei copil.

Pentru declararea regulilor de urmat într-o restricţie referenţială şi pentru a nu avea „surprize" neplăcute, se vor parcurge, în ordine, următorii paşi:

1. În fereastra de comandă se scrie şi se execută secvenţa de comenzi:
```
CLOSE TABLES ALL
OPEN DATABASE vinzari.dbc EXCLUSIVE
MODIFY DATABASE
```
2. Cu `Database Designer`-ul deschis, se alege din meniul sistem opţiunea `DATABASE→CLEAN UP DATABASE`;
3. Din acelaşi meniu se foloseşte opţiunea `DATABASE→EDIT REFERENTIAL INTEGRITY`, care va lansa formularul de editare a restricţiilor referenţiale (figura 2.30).

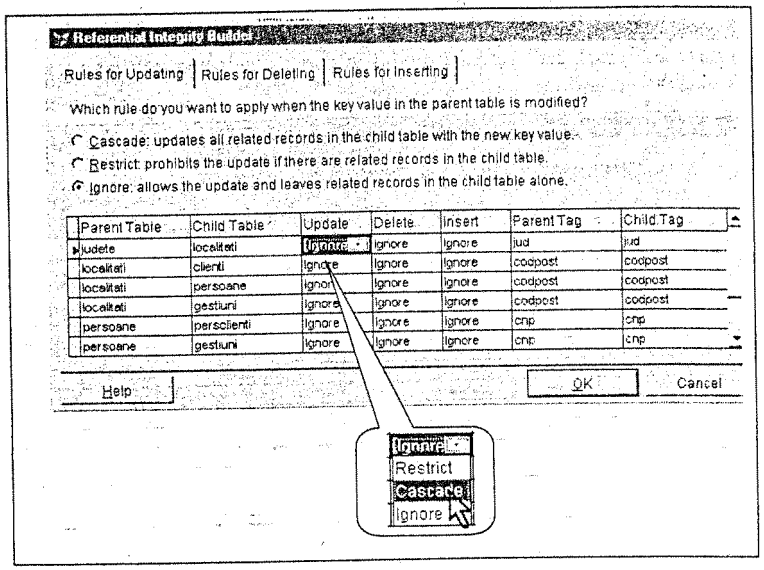


Figura 2.30. Construirea restricţiilor referenţiale

După cum se observă în figura 2.30, fiecare legătură permanentă declarată anterior între două tabele are o linie corespondentă în tabela de editare a restricţiilor referenţiale astfel: primele două coloane din stânga conţin numele tabelelor implicate în relaţie şi care dintre ele este părinte sau copil, ultimele două coloane conţin *numele indecşilor* de legătură, iar cele trei coloane din mijloc se referă la regulile instituite la inserarea, modificarea şi ştergerea de înregistrări în tabelele-părinte şi copil, operaţiuni care ar putea afecta integritatea referenţială. Fiecare dintre celulele acestor trei coloane ascunde un obiect de tip listă derulantă din care putem alege opţiunile dorite.

Cele trei reguli pe care le putem defini se referă la :

- modificarea cheii primare în tabela-părinte;
- ştergerea unei înregistrări în tabela-părinte;
- inserarea unei înregistrări sau modificarea valorii cheii străine în tabela-copil.

În mod normal, conform modelului relaţional, vom institui următoarele reguli:

- modificarea cheii primare în tabela-părinte trebuie să declanşeze modificarea automată a vechilor valori ale cheilor străine din tabelele-copil (aşa-numita modificare în cascadă);
- nu se pot şterge înregistrări din tabela-părinte dacă valorile cheii primare se regăsesc pe atributele ce constituie cheie străină în tabelele-copil;
- nu se pot introduce, în tabelele-copil, valori ale atributelor-cheie străină care să nu se regăsească în atributele-cheie primară din tabela-părinte.

Putem spune că toate cele trei reguli converg spre acelaşi scop comun şi anume: baza de date nu trebuie să permită să se ajungă în situaţia ca într-o tabelă-copil să existe valori pe atributele-cheie străină care să nu se regăsească pe atributele-cheie primară din tabela (tabelele) părinte. Figura 2.31 reflectă modul în care va arăta editorul restricţiilor referenţiale după stabilirea regulilor:

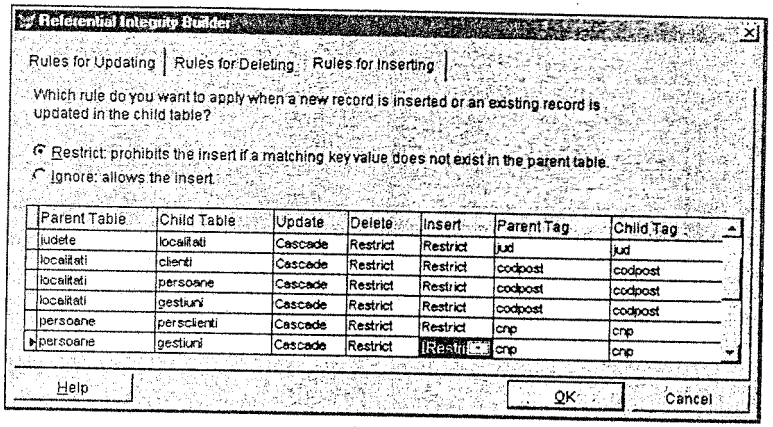


Figura 2.31. Editorul restricţiilor referenţiale după stabilirea regulilor

Practic, luând spre exemplificare tabela LOCALITATI, în modulul de editare a restricţiilor de integritate referenţială prezentat în figura 2.31 s-au instituit următoarele reguli:

- modificarea valorii codului poştal în tabela LOCALITATI atrage modificarea în cascadă a tuturor liniilor-copil (linii în care atributul `codpost` avea valoarea dinaintea modificării) în tabelele CLIENTI, PERSOANE şi GESTIUNI;
- modificarea valorii câmpului `jud` în tabela JUDETE atrage modificarea în cascadă a tuturor liniilor-copil în LOCALITATI;
- nu se şterg linii din tabela LOCALITATI şi JUDETE dacă există linii-copil corespondente în tabelele CLIENTI, PERSOANE, GESTIUNI, respectiv LOCALITATI (copil al tabelei JUDETE);

- nu se permite inserarea sau modificarea unei linii în tabelele CLIENTI, PERSOANE, GESTIUNI dacă valoarea cheii străine, `codpost`, nu există în tabela-părinte LOCALITATI;
- nu se permite inserarea sau modificarea unei linii în tabela LOCALITATI dacă valoarea cheii străine, `jud`, nu există în tabela-părinte JUDETE.

Nu mai rămâne decât de apăsat butonul OK, apoi să răspundem cu YES sau OK la cele două mesaje ce vor apărea înainte ca SGBD-ul să implementeze regulile respective şi, apoi, să testăm restricţiile generate.

Mesajul universal de eroare pe care îl produce motorul bazei de date atunci când o restricţie referenţială este încălcată este `TRIGGER FAILED`. Pentru a obţine un comportament mai „elegant" (mesajul de eroare să fie inteligibil utilizatorilor), există o altă modalitate de implementare a restricţiilor referenţiale, modalitate tratată pe larg în capitolul 6.

Figurile următoare reflectă testarea restricţiilor generate anterior.

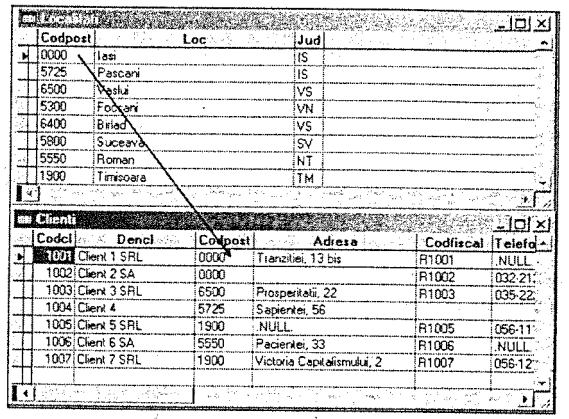


Figura 2.32. *Update*-ul în cascadă (codul laşiului : 6600 devine 0000)

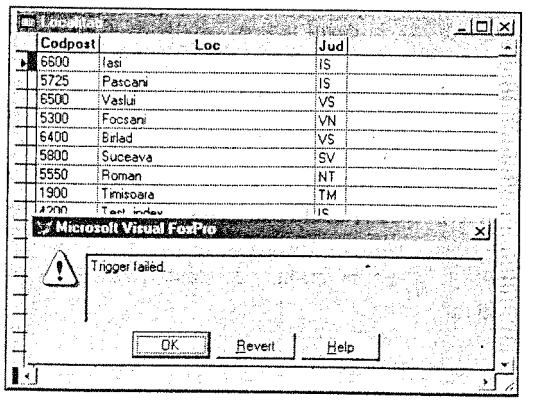


Figura 2.33. Mesaj de eroare la încercarea de a şterge o linie ce are linii-copil

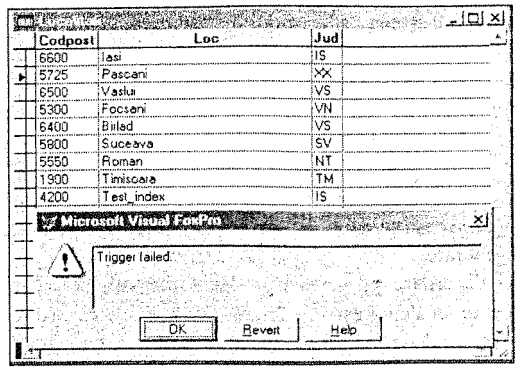


Figura 2.34. Mesaj de eroare la încercarea de introducere pe atributul `jud` a unei valori („XX") ce nu se regăseşte în tabela-părinte `JUDETE` pe acelaşi atribut

2.5.3. Valori implicite pentru atributele de tip cheie străină

În lipsa *buffering*-ului (detaliat în capitolele 9 şi 13), la inserarea unei linii în tabela-copil se verifică dacă valorile atributelor-cheie străină se regăsesc în tabelele-părinte. Cum aceste valori sunt vide, se va afişa un mesaj de eroare de tipul `Trigger failed`. Cea mai neinspirată soluţie este de a introduce în toate tabelele-părinte câte o înregistrare vidă, drept corespondent al noilor valori introduse pe liniile-copil.

Mult mai inspirată este însă precizarea unor valori implicite pe care să le ia atributele-copil la adăugarea unei linii noi în tabelele respective, valoare care sigur există în tabela-părinte şi care poate fi specificată prin sintaxa generală:

`numetabelă_părinte.atribut_de_legătura` (adică valoarea curentă de pe atributul de legătură din tabela-părinte[16]).

Atenţie! Această modalitate de tratare a valorilor implicite necesită obligatoriu *deschiderea tabelei-părinte* înainte de adăugarea de înregistrări în tabela-copil.

Spre exemplu (figura 2.35), în cazul tabelei LINIIFACT (`NrFact` şi `CodPr` – chei străine) paşii sunt următorii:

- se deschid tabelele PRODUSE şi FACTURI (prin comanda `USE` sau dublu-click pe fiecare tabelă în `Database Designer`);
- se lansează editorul structurii tabelei LINIIFACT (butonul `Modify` în `Project Manager` sau comanda `MODIFY STRUCTURE`);
- pentru câmpul `NrFact` (tabela LINIIFACT) se introduce în casuţa de text `Default value`: `facturi.nrfact`;
- pentru câmpul `CodPr` în casuţa de text `Default value` se tastează `produse.codpr`

16. Pentru detalii, vezi capitolul 3.

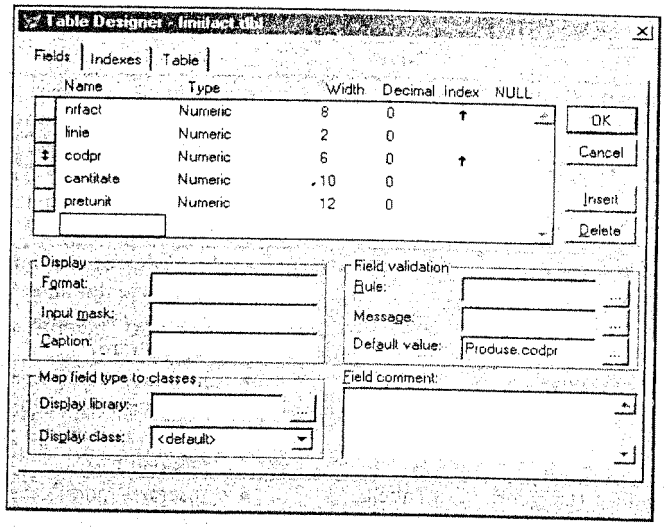


Figura 2.35. Valori implicite pe cheile străine

Atâta timp cât nu se mută pointerul pe altă înregistrare a tabelei-părinte, valoarea implicită în tabela-copil va fi întotdeauna aceeaşi, lucru ce poate duce uneori la încălcarea unicităţii cheii primare.

De exemplu, dacă tabela LINIIFACT ar fi avut drept cheie primară combinaţia (`NrFact`, `CodPr`), la adăugarea celei de-a doua linii într-o factură, fără modificarea prealabilă a pointerului din FACTURI şi/sau PRODUSE, s-ar fi încălcat restricţia de unicitate.

Pentru evitarea unor asemenea situaţii, există posibilitatea definirii, ca valoare implicită pe atributul în cauză, a unei funcţii-utilizator care să întoarcă întotdeauna o valoare „condiţionată" (în cazul expus mai sus, un cod de produs care să nu existe deja pe factura curentă). Despre o asemenea funcţie însă, vom discuta în capitolul 6.

2.5.4. Momentul activării restricţiilor unei baze de date

Nu putem încheia prezentarea noastră din acest capitol fără să facem referire şi la momentul declanşării diferitelor tipuri de validări (restricţii) definite pentru o tabelă anume.

În acest sens, putem surprinde următoarele situaţii posibile:

RESTRICŢIE		Moment declanşare
Tip	Nivel	
Cheie primară/candidat	Înregistrare	• adăugarea unei noi înregistrări (chiar în momentul adăugării) • modificarea unei înregistrări (în momentul mutării pointerului pe altă înregistrare)

RESTRICȚIE		Moment declanșare
Tip	Nivel	
Restricție la nivel de atribut	Câmp	• trecerea de la un atribut la altul (pe aceeași înregistrare sau pe înregistrări diferite), dar *numai dacă s-a modificat valoarea* pe câmpul restricționat
Restricție la nivel de înregistrare	Înregistrare	• modificarea unei înregistrări (în momentul mutării pointerului pe altă înregistrare)
Restricții referențiale	Tabelă	• inserare, modificare, ștergere înregistrări (în momentul mutării pointerului pe altă înregistrare)
Declanșatoare (triggere)	Tabelă	• inserare, modificare, ștergere înregistrări (în momentul mutării pointerului pe altă înregistrare)

Capitolul 3

Nucleul xBase. Actualizarea înregistrărilor

După crearea structurii bazei de date urmează etapa de utilizare efectivă a acesteia: popularea tabelelor cu înregistrări, modificarea valorilor unor atribute, ștergerea liniilor inutile sau eronate. De multe ori, modificarea unei înregistrări necesită și folosirea unui mecanism de acces rapid la respectiva linie.

3.1. Zone de lucru. Deschiderea tabelelor bazei de date sau a tabelelor libere

Accesul la înregistrările unei tabele Visual FoxPro, indiferent dacă este integrată într-o bază de date sau este independentă, se realizează prin intermediul unor structuri de memorie gestionate de SGBD. Acestea, deși au un rol similar recordset-urilor din Access sau Visual Basic, se numesc *zone de lucru*.

În FoxPro, primele 25 de zone de lucru sunt identificabile prin litere (A...Z) sau cifre (1...25). După această limită rămâne doar identificatorul numeric. Aceste zone de lucru sunt inițial libere (neocupate de seturile de înregistrări ale vreunei tabele) și pot fi ocupate de tabele obișnuite VFP (care corespund unui fișier `.dbf`) ori de tabele temporare, tip cursor sau tabelă derivată (*view*).

Prin urmare, pentru a lucra cu înregistrările unei tabele VFP (afișare, listare, modificare etc.), aceasta trebuie deschisă explicit. Astfel, tabelei deschise i se va asocia o zonă de lucru, care poate fi identificată fie prin numele ei, fie prin aliasul precizat în comanda `USE`:

```
USE <nume tabelă> IN <zonă de lucru> ALIAS <alias>
```

Dacă se omite clauza `ALIAS`, acesta va fi considerat implicit numele fișierului `.dbf`.

Ce se întâmplă dacă se încearcă deschiderea unei tabele într-o zonă de lucru ocupată deja? Fără nici un mesaj de avertizare, sistemul închide tabela deschisă în zona respectivă și deschide în locul ei noua tabelă. Acest comportament poate fi evitat prin folosirea clauzei `IN` și o zonă de lucru liberă. Pentru a verifica dacă o zonă de lucru este liberă sau nu, folosim funcția `USED()`, ce primește ca argument numele (aliasul) zonei de lucru în cauză. Dacă este liberă, funcția va întoarce valoarea `.F.(False)`. Implementarea unui mecanism în scopul evitării suprascrierii unei zone de lucru ocupate deja ar presupune o structură de control repetitivă, în care să fie invocată funcția amintită mai sus cât timp aceasta ar întoarce valoarea `.f.(False)`. Versiunile vizuale ale FoxPro vin însă cu un mecanism simplu și elegant prin utilizarea clauzei `IN` cu valoarea `0 (zero)`, astfel încât sistemul va căuta singur prima zonă liberă:

```
USE facturi IN 0
```

Ce se întâmplă dacă se încearcă deschiderea unei tabele deja deschise? De obicei, apare un mesaj de eroare `File in use (Error 3)`. Totuşi, incluzând în comanda `USE` clauza `AGAIN` şi folosind eventual şi un alias diferit, putem deschide una şi aceeaşi tabelă în mai multe zone de lucru. De exemplu:

```
USE facturi IN 1
USE facturi IN 2 ALIAS fct AGAIN
```

Orice actualizare a înregistrărilor uneia dintre instanţele tabelei va fi reflectată automat de toate celelalte instanţe deschise la un moment dat. Dacă nu se specifică un `ALIAS`, sistemul va alege singur unul care să nu se mai regăsească în nici una din zonele de lucru ocupate.

Secvenţa de program din listingul 3.1 deschide tabela FACTURI, verificând mai întâi dacă aceasta nu este deschisă deja. Dacă este deschisă, afişează un mesaj[17] utilizatorului şi, în funcţie de răspunsul acestuia, o deschide din nou sau nu.

Listing 3.1. Deschiderea unei tabele

```
IF !USED('Facturi')
    USE Facturi IN 0
ELSE
a = MESSAGEBOX("Tabela FACTURI este deja deschisă." +;
    "Doriti sa o mai deschideti odata intr-o alta zona de lucru ?", 4+32+256, "Confirmare")
IF a = 7
    USE Facturi IN 0 ALIAS Facturi_2
ENDIF
ENDIF
```

Modalitatea standard de verificare vizuală a tabelelor deschise la un moment dat într-o sesiune de lucru Visual FoxPro constă în selectarea opţiunii din meniul sistem: `WINDOW→DATASESION` sau utilizarea comenzii `SET` în fereastra de comandă (figura 3.1).

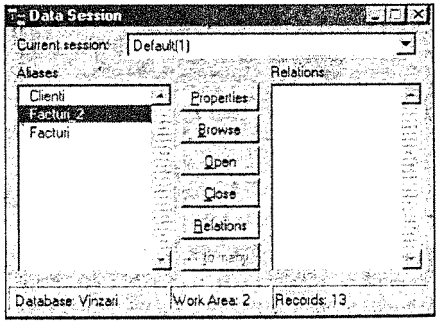


Figura 3.1. Tabelele deschise în sesiunea curentă

Deschiderea unei tabele nu este suficientă pentru a avea acces la înregistrările acesteia. Dintre toate zonele de lucru, cea asupra căreia au incidenţă directă comenzi simple ca `DISPLAY`, `BROWSE`, `REPLACE` se numeşte *zona de lucru curentă*. Prin urmare, următorul pas este selectarea zonei în care este deschisă tabela vizată, zonă care va deveni astfel curentă (în figura 3.1, zona curentă este `Facturi_2`).

17. Vezi parametrii funcţiei `MESSAGEBOX()` în documentaţia VFP.

De exemplu:

```
SELECT 1 sau SELECT facturi
LIST
```

Selectarea unei zone prin comenzi se reflectă automat în `Data Sesion` şi invers. Ordinea tabelelor în `Data Sesion` este cea alfabetică şi nu ordinea în care au fost deschise.

Din acest moment ne vom referi la tabela deschisă în zona curentă de lucru prin sintagma *tabelă curentă.*

Dacă nu se cunoşte exact numele tabelei care urmează a fi deschise, sau pur şi simplu nu se găseşte în directorul curent (vezi parametrul `Default Directory` din `Tools→Options→FileLocations`) sau în căile de căutare alternative (vezi parametrul `Search Path` din acelaşi submeniu), atunci se poate invoca comanda `USE` folosind în locul numelui tabelei caracterul „?":

```
USE ? IN 0
```

Se va deschide astfel o fereastră de dialog în care sunt afişate tabelele din baza de date curentă (dacă este deschisă) sau o fereastră tip `Open` (care se deschide şi din butonul `<Other...>` al dialogului anterior). Se poate naviga astfel până în directorul în care se găseşte fişierul corespunzător tabelei respective:

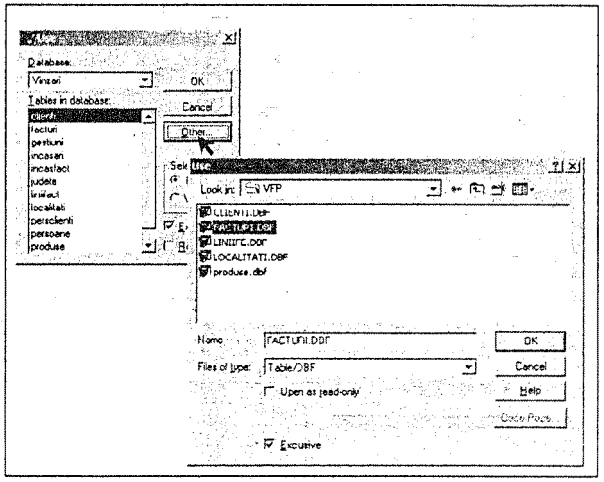


Figura 3.1 bis. Dialog pentru deschiderea unei tabele

Rezultă aşadar că procedura de deschidere a unei tabele independente este uşor diferită de cea pentru o tabelă inclusă în baza de date. Dacă se procedează absolut „disciplinat", pentru o tabelă „legată", ar trebui deschisă mai întâi baza de date printr-o comandă `OPEN DATABASE` şi apoi deschisă şi tabela:

```
OPEN DATA vinzari
USE facturi IN 0
```

Deschiderea tabelei din baza de date printr-o comandă `USE` implică deschiderea automată a **bazei de date** dacă fişierul ce corespunde dicţionarului bazei de date (*fişier. dbc*) se găseşte pe calea implicită de căutare.

Folosind Project Manager-ul, o tabelă poate fi deschisă astfel:

- dacă este vorba despre o tabelă inclusă în baza de date:
  1. selectarea nodului Data sau a paginii Data;
  2. expandarea nodului Database;
  3. expandarea nodului corespunzător bazei de date în care se află tabela (de exemplu, VINZARI);
  4. expandarea nodului Tables;
  5. selectarea tabelei vizate şi click pe butonul Browse sau Modify care, înainte de afişarea ferestrei Browse sau Modify Structure, deschide şi tabela în prima zonă liberă.
- dacă este vorba despre o tabelă independentă:
  1. selectarea nodului Data sau a paginii Data;
  2. expandarea nodului Free Tables;
  3. selectarea tabelei vizate şi click pe butonul Browse sau Modify.

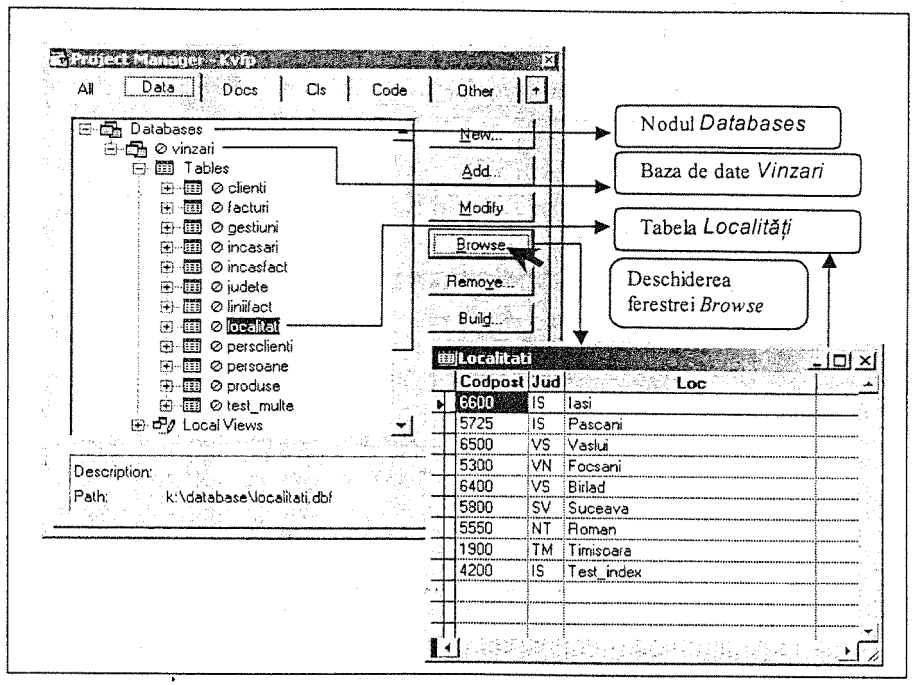


Figura 3.2. Deschiderea unei tabele din Project Manager

3.2. Noţiunea de pointer sau înregistrare curentă. Navigare

În Visual FoxPro, înregistrarea, din setul de înregistrări al unei tabele deschise, care este afectată direct printr-o comandă simplă gen DISPLAY, se numeşte *înregistrare curentă* – sau, altfel spus, ***pointerul tabelei*** *indică înregistrarea curentă*. Într-o fereastră tip

BROWSE, înregistrarea curentă este scoasă în evidenţă printr-un marcaj specific (vezi figura 3.3):

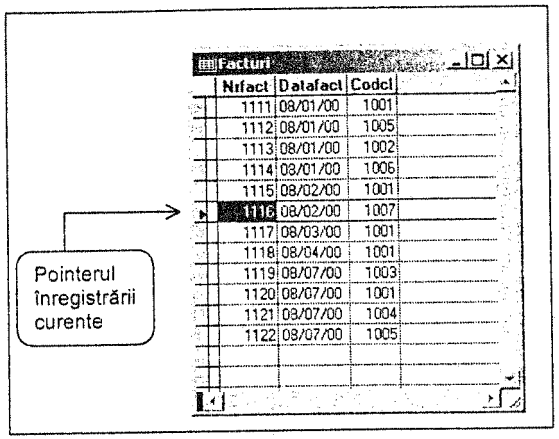


Figura 3.3. Pointerul înregistrării curente

Important! Valoarea unui anumit câmp de pe înregistrarea curentă se obţine utilizând sintaxa generală: Numetabelă.Câmp. Spre exemplu, conform figurii 3.3, afişarea codului clientului de pe înregistrarea curentă se poate face astfel:

```
? Facturi.Codcl
```

sau

```
MESSAGEBOX(STR(Facturi.Codcl))
```

Schimbarea sau mutarea pointerului de înregistrare se poate face prin comenzi de repoziţionare directă cum sunt GO, GOTO, SKIP sau prin repoziţionare indirectă folosind comanda LOCATE, SEEK sau funcţia SEEK().

Sintaxa comenzii GO sau GOTO poate fi redată astfel:

```
GOTO [RECORD] număr_înregistrare
 [IN zonă_de_lucru | IN alias_tabelă]
```

sau

```
GOTO TOP | BOTTOM [IN zonă_de_lucru | IN alias_tabelă]
```

Astfel:

- dacă se încearcă mutarea pointerului tabelei pe înregistrarea cu numărul 3 (în fişierul corespunzător de pe disc, .dbf, ***înregistrările au asociat un număr de identificare unic, alocat fizic în ordinea în care au fost introduse***), se procedează ca în listing 3.2.

Listing 3.2. Poziţionarea directă pe altă înregistrare

```
USE FACTURI IN 0
SELECT FACTURI
GO RECORD 3
*** Sau
GO 3
```

Precizare. Numărul intern al fiecărei înregistrări poate fi obținut cu ajutorul funcției `RECNO()`, iar numărul total de înregistrări al unei tabele, cu funcția `RECCOUNT()`.

- dacă se dorește a se ajunge pe prima înregistrare indiferent unde este plasat pointerul tabelei, se procedează astfel: `GO TOP`
- dacă se dorește deplasarea pointerului tabelei pe ultima înregistrare, este necesară comanda: `GO BOTTOM`

Sintaza comenzii `SKIP` poate fi redată astfel:

`SKIP [număr de înregistrări] [IN zonă_de_lucru | IN alias_tabelă]`

Ca urmare:

- dacă se dorește deplasarea înainte peste 3 înregistrări, comanda `SKIP` va fi invocată astfel:

```
SKIP 3
```

- dacă se dorește deplasarea înapoi cu 2 înregistrări, va trebui procedat în felul următor:

```
SKIP -2
```

Dacă se dorește să se afle numărul înregistrării curente după ce s-au făcut mai multe manevre de repoziționare, se poate folosi funcția `RECNO()`:

```
?RECNO()
```

De asemenea, deplasarea într-o tabelă se poate face și în mod asistat, folosind opțiunile din meniul `Table` (vezi figura 3.4):

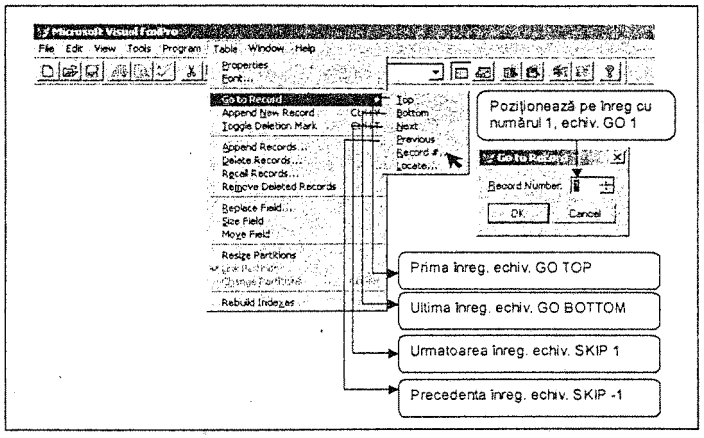


Figura 3.4. Poziționarea în mod asistat în tabela curentă

Există situații în care prin manevre de repoziționare, pointerul înregistrării curente ar putea să depășească domeniul de înregistrări. De exemplu:

- tabela conține 10 înregistrări și se execută o comandă `SKIP 30`. Ca urmare, pointerul tabelei este trimis într-o zonă aflată dincolo de limita numărului de înregistrări, și anume în zona ***end of file*** (adică după ultima înregistrare). Pentru a se verifica acest lucru se poate utiliza funcția `EOF()`: dacă se obține valoare `.T.(True)` înseamnă că s-a depășit numărul de înregistrări; dacă se obține valoare `.F.(False)` înseamnă că pointerul tabelei se găsește undeva pe o anume înregistrare;

- tabela conține 15 înregistrări și se execută o comandă `SKIP -17`. Ca urmare, pointerul tabelei este trimis într-o zonă aflată dincolo de limita numărului de înregistrări, și anume în zona ***begin of file*** (adică înainte de prima înregistrare). Pentru a se verifica acest lucru, se poate testa funcția `BOF()`: dacă se obține valoare `.T.(True)` înseamnă că s-a depășit numărul de înregistrări, dacă se obține valoare `.F.(False)` înseamnă că pointerul tabelei se găsește undeva pe o anumită înregistrare a tabelei.

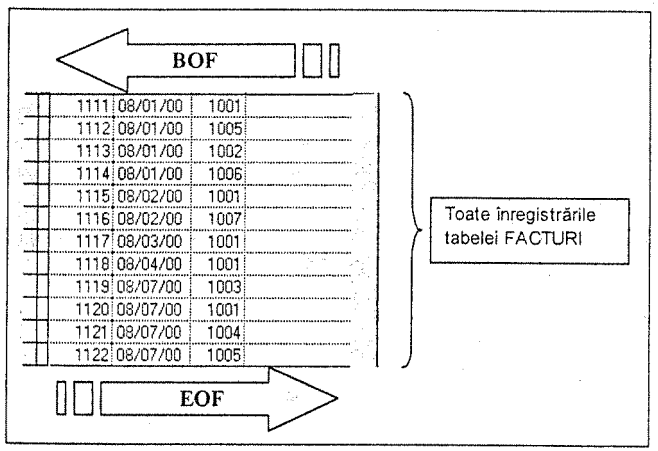


Figura 3.5. Zonele în afara domeniului de înregistrări

3.3. Acces la înregistrări în modul indexat și neindexat

Deplasarea pointerului într-o tabelă se poate face și prin localizarea înregistrărilor după o anumită condiție. Una dintre comenzile cu care se poate realiza acest lucru este `LOCATE`, care parcurge tabela, înregistrare cu înregistrare, până o găsește pe ***prima care îndeplinește condiția și se poziționează pe ea***. Comanda are următoarea sintaxă:

```
LOCATE FOR <expresie logică1> [<domeniu>]
    [WHILE <expresie logică2>] [NOOPTIMIZE]
```

De exemplu, pentru a se localiza facturile din data de 8 august 2001, se poate proceda astfel:

Listing 3.3. Căutare valori folosind `LOCATE`

```
USE Facturi IN 0
SELECT Facturi
SET DATE TO DMY
```

```
LOCATE FOR DataFact = {01/08/2001} && data calendaristică se specifică între { } şi trebuie
                                     && respectat formatul specificat prin SET DATE
?FOUND()
DISPLAY
CONTINUE
?FOUND()
DISPLAY
```

Funcţia FOUND() întoarce valoarea de adevăr .T. (True) dacă ultima căutare a avut succes. *Dacă ultima căutare nu a avut succes* (funcţia a întors valoarea .F.), atunci *pointerul tabelei se află întotdeauna în zona EOF.*

Comanda CONTINUE declanşează o nouă secvenţă de căutare, de la înregistrarea pe care s-a plasat pointerul tabelei ca urmare a ultimei comenzi LOCATE spre sfârşitul fişierului.

Comanda DISPLAY afişează înregistrarea curentă.

În modul asistat, acelaşi lucru poate fi realizat (bineînţeles, pentru tabela curentă) folosind opţiunea de meniu Table→Go to record→Locate (vezi figura 3.6).

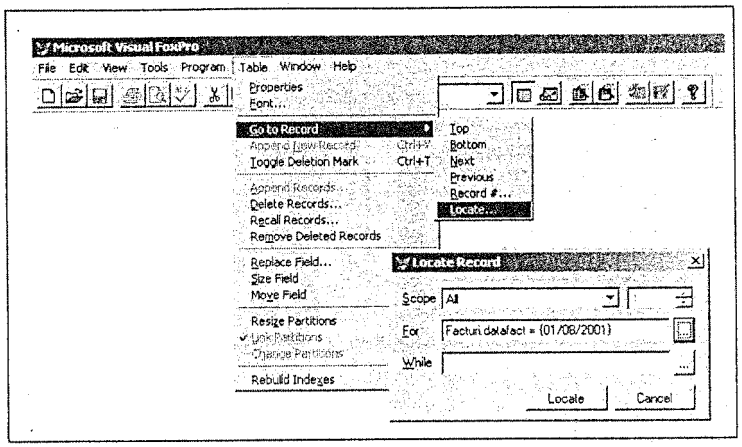


Figura 3.6. Localizarea unei înregistrări după o expresie, în modul asistat

Căutarea unei înregistrări folosind comanda LOCATE este destul de flexibilă în ceea ce priveşte criteriul specificat prin clauza FOR, însă pentru tabele mai „voluminoase" (câteva mii de înregistrări) se poate dovedi destul de anevoioasă. Acest fapt se datorează modului de parcurgere secvenţială a înregistrărilor care generează performanţe slabe în ceea ce priveşte timpii de execuţie.

De obicei, în mai toate SGBD-urile relaţionale, ameliorarea vitezei de acces la înregistrările unei tabele se face pe baza indecşilor. În Visual FoxPro, comanda SEEK reprezintă o alternativă de căutare care invocă indecşii. De asemenea, se poate utiliza funcţia SEEK(). Prin urmare, pentru a efectua o astfel de operaţiune, indexul care va fi invocat trebuie să existe, să fie deschis şi să fie curent. În cazul indecşilor compuşi structurali (cel mai des utilizaţi în VFP), aceştia se deschid o dată cu tabela pentru care sunt definiţi, urmând a fi realizată doar activarea cheii corespunzătoare. Activarea indexului dorit se realizează de obicei cu ajutorul comenzii SET ORDER TO numeindex. Ca şi

comanda LOCATE, SEEK ***se va poziţiona pe prima înregistrare care îndeplineşte condiţia***, dar căutarea se realizează în fişierul ce conţine indecşii (.cdx) şi nu în tabelă.

Sintaxa comenzii SEEK arată cam în felul următor:

```
SEEK <expresie>
    [ORDER <expN> | <fişier.idx>]
    | [TAG <nume cheie> [OF <fişier.cdx>]
    [ASCENDING | DESCENDING]]
    [IN <zonă de lucru> | <alias>]
```

Să considerăm tabela FACTURI care are definiţi indecşii NRFACT după câmpul nrfact, CODCL după câmpul codcl şi GESTIUNE după câmpul gestiune (vezi capitolul 2 pentru structura şi înregistrările tabelelor). Pentru localizarea primei facturi a clientului ce are codul 1004 se poate proceda astfel:

- Se deschide tabela şi se activează cheia corespunzătoare

```
USE facturi IN 0 ORDER TAG codcl
SELECT facturi
sau
USE facturi IN 0
SELECT facturi
SET ORDER TO TAG codcl
```

- Se caută prima factură a clientului 1004

```
SEEK 1004
```

- Dacă a avut succes căutarea, se afişează înregistrarea curentă (DISPLAY) sau se deschide tabela în mod Browse (comanda BROWSE) şi se urmăreşte poziţia pointerului de înregistrare:

```
? FOUND()
BROWSE
```

În cazul în care cheia de căutare nu este activată, comanda SEEK ar trebui folosită în felul următor:

```
SEEK 1004 ORDER TAG codcl
```

Există anumite probleme legate de exactitatea şirului căutat. De exemplu, să presupunem că în tabela CLIENTI există cheia (tag-ul) DENCL după câmpul dencl. Însă anumiţi clienţi sunt consemnaţi ca 'SRL', 'SA', iar unii nu au nici o astfel de precizare la sfârşitul numelui. Dorim să localizăm înregistrarea clientului 'Client 4', dar nu ştim nici dacă este consemnat ca societate pe acţiuni ('Client 4 SA'), nici ca societate cu răspundere limitată ('Client 4 SRL'). Prin urmare, căutăm o ***valoare asemănătoare***, fără să ştim dacă va fi similară sau identică cu şirul de căutare. În cazul în care opţiunea de mediu[18] SET EXACT este ON, atunci e foarte posibil să nu avem succes dacă numele clientului căutat este 'Client 4 SRL' sau 'Client 4 SA'. În astfel de cazuri este preferabil ca SET EXACT să fie OFF, iar SET NEAR – ON. Interpretarea rezultatului unei asemenea căutări este explicată în tabelul 3.1.

18. Vezi capitolul 1, pentru comenzile de tip SET.

Tabelul 3.1. Interpretarea rezultatului căutării după o cheie indexată

EOF()	FOUND()	Rezultatul căutării
.F.	.T.	A fost găsită o înregistrare cu o valoare identică cu cea a şirului de căutare
.F.	.F.	Nu s-a găsit nici o înregistrare identică, însă pointerul tabelei se găseşte pe o înregistrare cu o valoare comparabilă – valoarea căutată este un subşir al valorii curente
.T.	.F.	Nu s-a găsit nici o valoare asemănătoare sau identică

Funcţia SEEK() are o funcţionalitate asemănătoare, adică va trimite pointerul de înregistrarea ce conţine valoarea căutată şi va returna:

- .T. (True) dacă găseşte o valoare identică;
- .F. (False) dacă nu găseşte o valoare identică.

Sintaxa acestei funcţii este următoarea:

```
SEEK(<expresie> [, <zonă de lucru> | <alias tabelă>
  [, <expN> | <fişier.idx> | <nume cheie>]])
```

Pentru exemplele anterioare, alternativa cu funcţia SEEK() ar fi următoarea:

```
? SEEK(1004, 'FACTURI', 'CODCL')
BROWSE
sau,
? SEEK('Client 5', 'CLIENTI', 'DENCL')
DISPLAY
```

Trebuie să precizăm faptul că funcţia SEEK() (sau comanda corespunzătoare) constituie întotdeauna cea mai bună alternativă pentru verificarea existenţei unei valori anume într-o tabelă datorită vitezei sporite de execuţie.

3.4. Adăugarea şi editarea înregistrărilor folosind modul asistat şi comenzi xBase

Editarea înregistrărilor bazei de date poate fi realizată în două moduri:

- Modul de lucru asistat (cu ajutorul interfeţei furnizate de SGBD);
- Modul de lucru comandă (prin utilizarea comenzilor specifice, xBase sau SQL).

3.4.1. Editarea înregistrărilor în mod asistat

Operaţiunea de actualizare presupune mai întâi deschiderea tabelei şi selectarea ei ca tabelă curentă (eventual deschiderea în mod Browse) şi apoi utilizarea opţiunilor submeniului sistem: TABLE (figura 3.7).

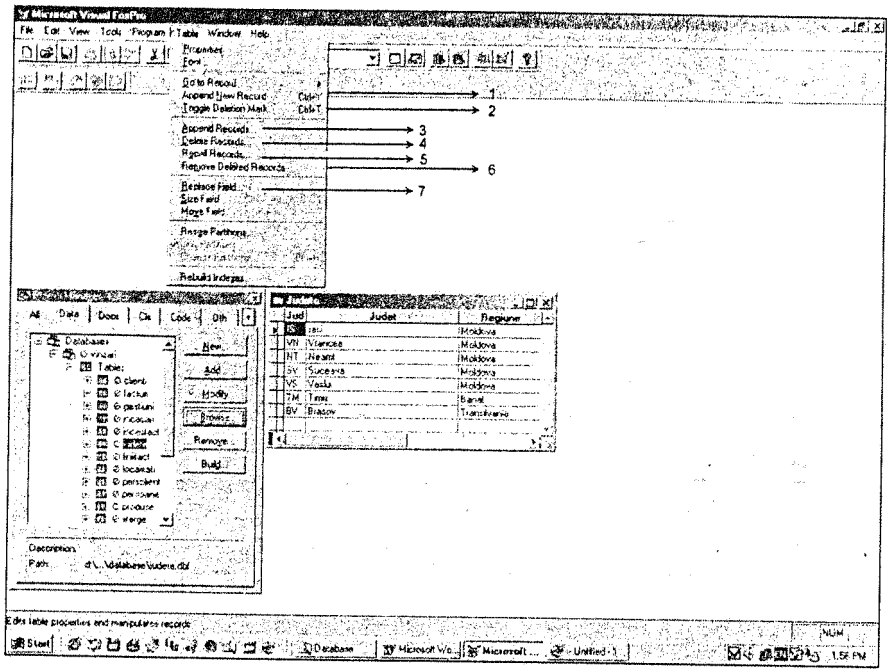


Figura 3.7. Opţiunile submeniului TABLE pentru actualizarea înregistrărilor tabelei curente în mod asistat

Aşadar, pentru ***adăugarea noilor înregistrări*** într-o tabelă folosim opţiunile 1 şi 3 astfel:

Opţiunea 1, Append New Record, adaugă o linie (înregistrare) nouă în tabela curentă. Combinaţia de taste Ctrl+Y (ca modalitate de apelare directă a opţiunii) este cea mai utilizată metodă de adăugare de înregistrări în mod interactiv. Nu trebuie să uităm că, pentru tabelele-copil, este absolut necesară definirea valorilor implicite pentru atributele care constituie chei străine (vezi capitolul 2).

Append Records... *(opţiunea 3)* – permite adăugarea „în bloc" a unor înregistrări noi din alte tabele (cu aceeaşi structură) sau din fişiere de alt tip (text, Excel, Paradox etc.). Bineînţeles că noile înregistrări trebuie să îndeplinească toate condiţiile impuse de restricţiile definite pentru tabela respectivă.

În ceea ce priveşte adăugarea de înregistrări din alte tipuri de fişiere, vom insista asupra preluării de date din fişiere de tip Excel şi de tip text, care trebuie să îndeplinească anumite condiţii, după cum urmează:

- în cazul *fişierelor Excel* (figura 3.8), datele trebuie aranjate în foaia de calcul începând obligatoriu cu prima coloană (A), fără antet (titlul coloanelor), SGBD-ul preluându-le în tabela curentă în ordinea în care apar, de la stânga la dreapta;

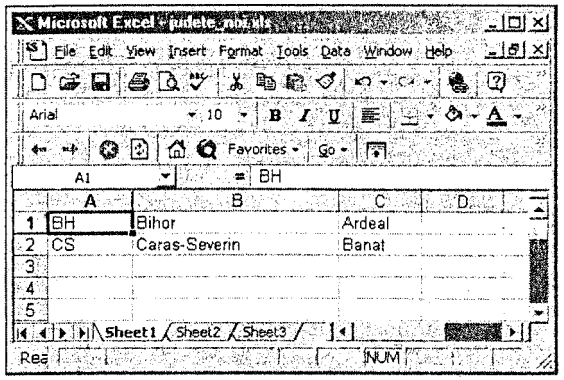


Figura 3.8. Foaie de calcul Excel ce urmează a fi preluată în tabela JUDETE

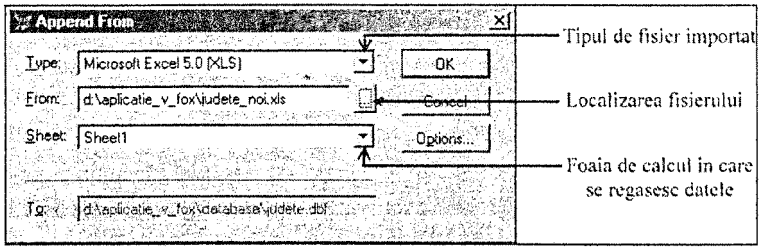


Figura 3.9. Utilizarea opțiunii APPEND RECORDS... pentru adăugare de noi înregistrări (în tabela JUDETE) importate dintr-o foaie de calcul Excel

- în cazul fișierelor de tip text (figura 3.10) avem nevoie de un delimitator al coloanelor, delimitator care poate fi marcajul TAB sau orice alt caracter care nu se regăsește și în secvențele de text ce vor fi importate. Spre exemplificare, vom considera virgula ca delimitator de câmpuri în fișierul Judete_noi.txt (figura 3.11).

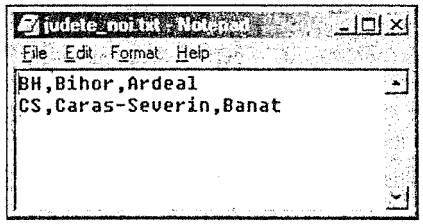


Figura 3.10. Fișier text ce urmează a fi importat în tabela JUDETE

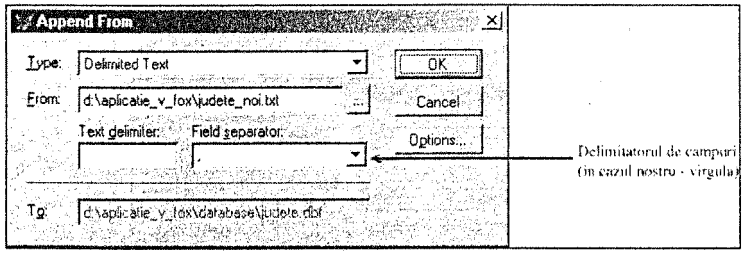


Figura 3.11. Utilizarea opțiunii APPEND RECORDS... pentru adăugare de noi înregistrări (în tabela JUDETE) importate din fișierul text Judete_noi.txt

Pentru ***modificarea înregistrărilor*** unei tabele avem *opțiunea 7* (conform figurii 3.7): REPLACE FIELD... Aceasta oferă utilizatorului posibilitatea de a specifica atributul (câmpul) ce va fi modificat, valoarea nouă ce va înlocui valorile curente ale câmpului respectiv, precum și înregistrările ce vor fi afectate de modificare (figura 3.12).

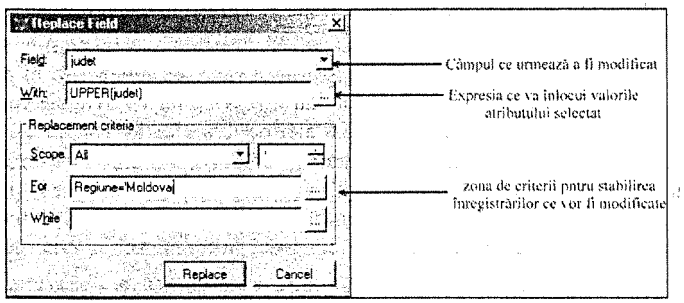


Figura 3.12. Modificarea valorilor atributelor în mod asistat

În ceea ce privește *zona de criterii*, facem următoarele *precizări general valabile atât pentru operațiuni de modificare, cât și pentru operațiuni de ștergere*, fie că se realizează în mod comandă sau asistat:

Clauza SCOPE determină scopul actualizării (care înregistrări vor fi afectate de modificare) și poate avea următoarele valori:

- All – toate înregistrările tabelei ;
- Next – următoarele *n* înregistrări începând cu poziția curentă a pointerului ;
- Record – doar înregistrarea specificată prin numărul ei ;
- Rest – restul înregistrărilor pornind din poziția curentă a cursorului.

Clauza FOR permite specificarea unei condiții astfel încât numai înregistrările ce o îndeplinesc vor fi afectate de modificare

Clauza WHILE permite și specificarea unei condiții atfel încât modificarea va avea loc atât timp cât înregistrările îndeplinesc respectiva condiție, parcurgându-se tabela de sus în jos, din poziția curentă a pointerului. La prima înregistrare care nu mai îndeplinește condiția dată, actualizarea este oprită.

Pentru cazul luat drept exemplu în figura 3.12, rezultatul se concretizează în transformarea în litere mari a numelor județelor din Moldova (figura 3.13).

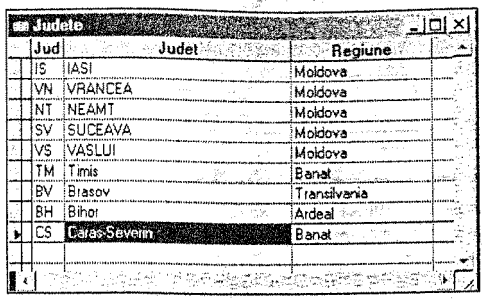


Figura 3.13. Tabela JUDETE actualizată

În sfârșit, ***ștergerea înregistrărilor*** în Visual FoxPro presupune parcurgerea, în ordine, a două faze: ștergerea logică și ștergerea fizică sau anularea marcajelor de ștergere. Astfel:

- *Ștergere logică* – reprezintă primul pas în ștergerea înregistrărilor și determină apariția unui marcaj în dreptul înregistrării în cauză (figura 3.14). În cadrul acestui pas are loc și declanșarea triggerelor sau a restricțiilor referențiale care vor valida sau nu operația de ștergere. Utilizăm aici opțiunile 2 și 4 (figura 3.7) – `Toggle Deletion Mark` (pentru înregistrarea curentă) și `Delete Records` (pentru marcarea mai multor înregistrări conform unei scheme) – sau pur și simplu dăm un click de mouse (valabil numai pentru tabela dechisă în mod Browse) în zona marcajului de ștergere din dreptul înregistrării pe care dorim să o ștergem.
- *Ștergere fizică* – determină îndepărtarea iremediabilă a înregistrărilor marcate pentru ștergere și se poate realiza cu opțiunea 6 (`Remove Deleted Records`). Acest pas necesită **deschiderea tabelei în mod exclusiv** (pentru lămuriri asupra lucrului în rețea, vezi capitolul 13), operațiunea nefiind posibilă pe o tabelă deschisă în mod partajat.
- *Anularea marcajelor de ștergere* ar putea constitui un pas alternativ al ștergerii definitive și se realizează fie utilizând opțiunea 5 (`Recall Records...`), fie printr-un click de mouse pe marcajul de ștergere al înregistrării în cauză.

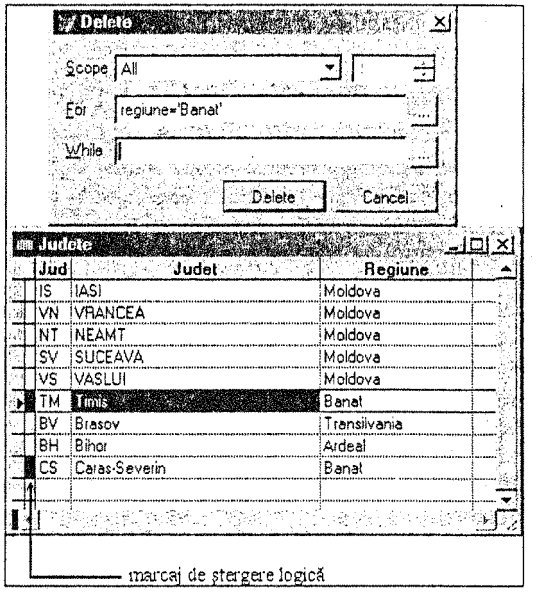


Figura 3.14. Ștergerea logică a înregistrărilor

3.4.2. Comenzi xBase pentru actualizarea directă a înregistrărilor

Toate operațiunile de actualizare despre care am discutat până acum pot fi realizate, bineînțeles, și prin comenzi specifice FoxPro sau instrucțiuni SQL (despre care vom discuta în capitolul 5).

Astfel, pentru adăgare de înregistrări utilizăm comanda generală `APPEND`, care deschide o fereastră ce conține câmpurile tabelei curente pentru adăugarea unei înregistrări noi; apoi `APPEND BLANK` – adaugă o înregistrare vidă (toate câmpurile vor avea valori vide, mai puțin cele care au definite valori implicite).

Extrem de utilă pentru importul de date (vezi și capitolul 4) este comanda `APPEND FROM`, asupra căreia insistăm doar în privința următoarelor aspecte:

- `APPEND FROM <nume_tabela.dbf>` – adaugă în tabela curentă înregistrările din tabela specificată. Tabela din care adăugăm trebuie să aibă aceleași câmpuri (mai ales ca nume), dar nu neapărat în aceeași ordine. Atributele care nu se potrivesc după nume sau tip sunt ignorate;
- `APPEND FROM ARRAY <nume_tablou>` – adaugă elementele unui tablou (în general o matrice) în tabela curentă, în ordinea naturală a coloanelor și liniilor;
- `APPEND FROM DBF("nume_cursor")` – adaugă în tabela curentă înregistrările din cursorul (tabelă temporară) specificat. De asemenea, câmpurile ce nu se potrivesc după nume și/sau tip sunt ignorate.

Trebuie menționat aici un aspect foarte important în ceea ce privește adăugarea de înregistrări noi, și anume: câmpurile de tip general (pentru imagini, sunete etc.). Singura

modalitate de inserare a unei imagini pe un astfel de câmp este utilizarea comenzii APPEND GENERAL, *după ce ne-am poziționat pe înregistrarea dorită*, cu sintaxa:

```
APPEND GENERAL NumeAtributTipGeneral [ FROM numefişier ]
                                     [ LINK ]
```

Utilizarea clauzei LINK determină crearea numai a unei legături între fişier şi înregistrarea curentă, fără a genera inserarea efectivă a obiectului în tabelă. Este recomandată această modalitate de lucru pentru că nu duce la sporirea excesivă a dimensiunilor tabelei, pe de o parte, iar pe de altă parte, nu necesită operaţiuni de actualizare ulterioare în cazul în care fişierul-sursă se modifică.

Spre exemplu, pentru inserarea unei imagini în tabela PRODUSE pe câmpul general Imagine pentru *produsul 2* din categoaria *bere* vom proceda astfel (figura 3.15):

- deschidem tabela în modul Browse şi ne poziţionăm pe înregistrarea a doua;
- pornind de la presupunerea că imaginea dorită se află în subdirectorul „Imagini", introducem în fereastra de comandă secvenţa:

```
APPEND GENERAL Imagine FROM imagini\bergman.gif LINK
```

După cum se observă, nu am utilizat calea completă spre fişierul bergman.gif presupunând că directorul Imagini este subdirector al directorului principal Aplicatie_v_fox, iar comanda SET DEFAULT TO D:\APLICATIE_v_FOX a fost utilizată anterior.

Pentru ştergerea unui obiect dintr-un câmp general folosim:

```
APPEND GENERAL NumeAtributGeneral, fără nici un alt parametru:
```

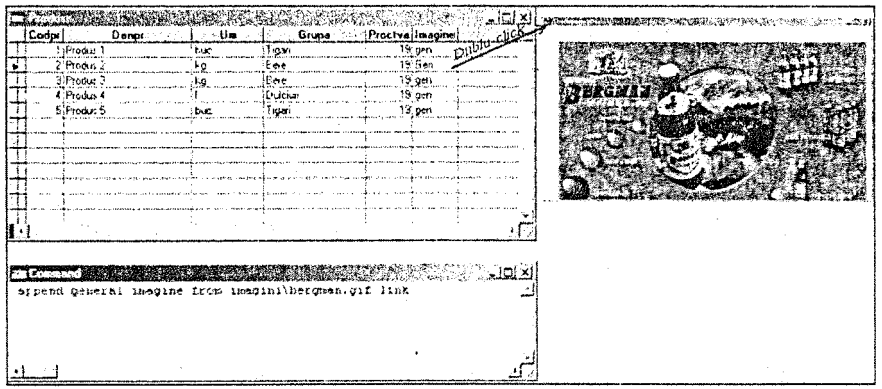


Figura 3.15. Adăugarea unui obiect (imagine în cazul de faţă) într-un câmp general şi vizualizarea lui

Pentru modificarea înregistrărilor bazei de date utilizăm comanda:

```
REPLACE NumeCâmp WITH  Expresie
  [Scop] [FOR Conditie1] [WHILE Conditie2]
```

care determină înlocuirea valorilor atributului specificat (din tabela curentă) cu noile valori, pentru toate înregistrările specificate prin clauzele Scope, For şi While. Utilizarea fără nici una din clauze duce la modificarea înregistrării curente. Spre exemplu, actualizarea datei tuturor facturilor trimise clientului *1001* poate fi realizată astfel:


```
SET DATE TO DMY
REPLACE Datafact WITH CTOD[19]("01/09/2000") FOR codcl=1001
```

Pentru ştergerea înregistrărilor utilizăm, în general, un grup de comenzi datorită modului specific de ştergere în două faze despre care am discutat mai sus (ştergerea logică şi fizică):

DELETE [Scop] [FOR Conditie1] [WHILE Conditie2] – *marchează* pentru ştergere (ştergere logică) înregistrarea curentă din tabela curentă sau înregistrările specificate prin scop şi/sau care îndeplinesc condiţiile din clauzele For sau While;

Înregistrările marcate pentru ştergere pot fi sau nu vizibile (spre ex.: într-o fereastră Browse), în funcţie de starea comenzii SET DELETED, astfel:

SET DELETED ON – înregistrările marcate pentru ştergere nu sunt vizibile şi nici accesibile anumitor tipuri de comenzi specifice VFP;

SET DELETED OFF – înregistrările marcate pentru ştergere sunt vizibile şi accesibile.

Indecşii primari iau în considerare („văd") înregistrările marcate pentru ştergere indiferent de setarea comenzii de mai sus. Nu acelaşi lucru este valabil pentru instrucţiunile SQL.

PACK – *şterge definitiv* (fizic) înregistrările marcate pentru ştergere din tabela curentă. **Nu poate fi utilizat în cazul tabelelor deschise în mod partajat (*shared*) sau în tranzacţii.**

RECALL [Scop] [FOR Conditie1] [WHILE Conditie2] – anulează marcajul de ştergere pentru înregistrarea curentă din tabela curentă sau înregistrările specificate prin scop şi/sau care îndeplinesc condiţiile din clauzele For sau While;

19. CTOD ('şir_caractere') - transformă şirul de caractere în format de tip dată calendaristică. Şirul trebuie să respecte formatul curent al datei (SET DATE TO DMY).

Capitolul 4

Programarea în Visual FoxPro

Pe lângă motorul de baze de date, Visual FoxPro oferă un limbaj de programare propriu destul de puternic, ce îmbină perfect stilul de programare clasic (procedural) cu programarea orientată-obiect, oferind suficiente instrumente de dezvoltare rapidă a unor aplicații puternice și flexibile. În acest capitol vom prezenta modelul de programare structurată, urmând ca în capitolul 11 să abordăm caracteristicile programării orientate-obiect.

4.1. Scurtă introducere în teoria programării

Programarea calculatoarelor este o disciplină destul de nouă, dar suportul său matematic este ceva mai vechi. În teoria programării calculatoarelor se specifică mai mulți pași în crearea unui program care să rezolve o anumită problemă. Acești pași merg de la formularea și specificarea problemei la definirea unei soluții, la implementare, testare și documentare și, în fine, la evaluarea soluției. Dacă anumite aspecte ale problemei desprinse din analiza problemei se pot reprezenta formal, este bine să se procedeze astfel, deoarece o problemă, o dată formalizată, este mai ușor de analizat prin prisma existenței unui mod de rezolvare (în cazul nostru, acest mod de rezolvare se concretizează în programul informatic).

Dacă problema noastră are un model matematic, pasul următor constă în găsirea soluției sub forma *algoritmului*. Vom spune, pe scurt, că un algoritm este o suită bine definită de *instrucțiuni* „clare" care se pot executa „cu o cantitate finită de efort și într-o perioadă finită de timp" și care „duc întotdeauna la obținerea unui rezultat".

În tehnica programării calculatoarelor, algoritmul acționează întotdeauna asupra unor *date de intrare*, ducând la obținerea *datelor de ieșire*. Într-un algoritm, aceste date se împart în *variabile* și *constante*, după modul în care își modifică sau nu valoarea pe parcursul execuției algoritmului. Dacă datele de intrare sunt numerice, atunci prelucrările pe care le va realiza algoritmul sunt prelucrări numerice; dacă e vorba de texte, prelucrările vor fi simbolice. Cât despre instrucțiune, aceasta este o operație care, cel puțin teoretic, nu poate fi descompusă în operații de nivel mai mic fără a-și pierde sensul. Astfel, adunarea nu poate fi descompusă, dar înmulțirea poate fi considerată o suită de adunări repetate. Practic, în informatică, elementare sunt numai instrucțiunile „turnate" în circuitele procesorului la proiectarea acestuia, dar programatorii s-au obișnuit să numească instrucțiuni și operații de un nivel ceva mai mare, ca de exemplu suita de operații necesare transferului unui text din memoria internă pe ecranul monitorului. Unii vorbitori (și scriitori) folosesc noțiunile *comandă* și *instrucțiune* ca sinonime, deși acestea nu sunt chiar astfel. Așadar, o comandă are o sferă mai largă decât o instrucțiune. De exemplu, o comandă de creare a unei tabele din Visual FoxPro sau o comandă de căutare a unor date într-o tabelă lansează de fapt un întreg algoritm. La fel se întâmplă și cu o comandă a sistemului de operare (de exemplu COPY, care, deși acest lucru nu este evident, există chiar și în Windows).

Într-un algoritm avem următoarele *tipuri de instrucțiuni*:

- de atribuire;
- de calcul;
- de decizie;
- de intrare-ieșire.

4.2. Structuri de control

Instrucțiunile algoritmului urmează căi de execuție numite *structuri de control*. Am folosit aici pluralul deoarece, în funcție de anumite condiții, prelucrarea datelor poate lua căi (ramificații) diferite. Structurile de control fundamentale sunt *secvența*, *selecția* și *iterația* (vezi figura 4.1).

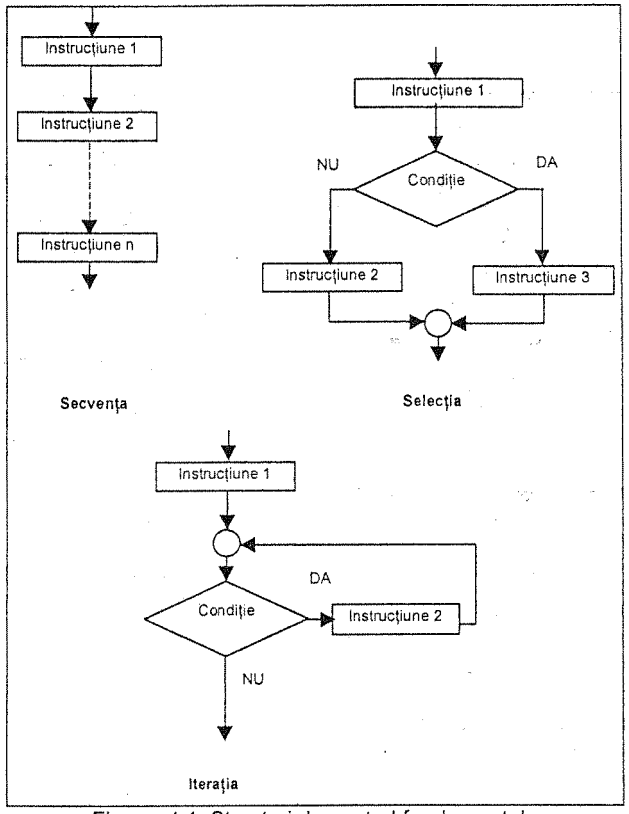


Figura 4.1. Structuri de control fundamentale

În 1965, Boehm și Jacopini au fundamentat teorema care le poartă numele, conform căreia orice algoritm poate fi reprezentat prin combinarea celor trei tipuri de structuri de control fundamentale.

Aşadar, în programarea structurată, avem următoarele posibilităţi:

- instrucţiunile se execută secvenţial în ordinea în care apar (de sus în jos), până la ultima (secvenţa);
- în funcţie de evaluarea unei condiţii la un moment dat, se execută un grup de instrucţiuni sau altul (selecţia sau structurile alternative). Trebuie precizat aici că putem avea structuri alternative cu o ramură vidă, ceea ce vrea să însemne că, în funcţie de evaluarea unei condiţii, se poate executa sau nu un anumit grup de instrucţiuni;
- un grup de instrucţiuni se execută de un anumit număr (finit) de ori, cunoscut dinainte sau cât timp o condiţie este îndeplinită (iteraţia sau structurile repetitive).

În practică, din iteraţie au derivat alte două tipuri de structuri de control (figura 4.2):

- structuri repetitive condiţionate anterior, caracterizate prin faptul că mai întâi se verifică o condiţie şi, dacă e adevărată, se execută o secvenţă de instrucţiuni, după care bucla se reia ;
- structuri repetitive condiţionate posterior, caracterizate prin faptul că mai întâi se execută o secvenţă de instrucţiuni, apoi se verifică o condiţie şi, dacă e adevărată, bucla se reia.

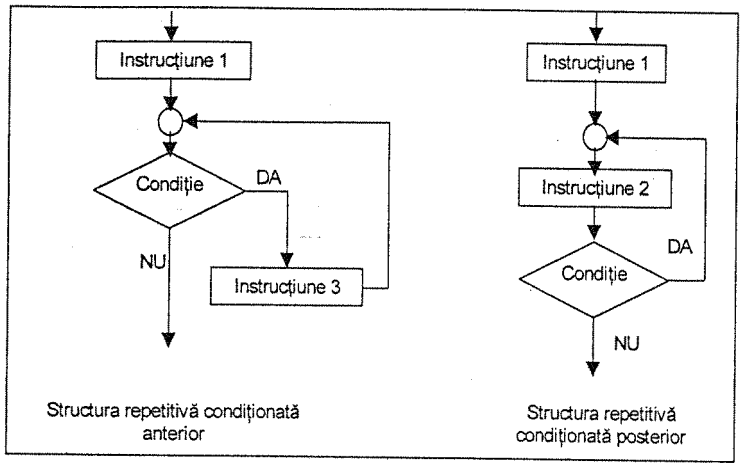


Figura 4.2. Subtipurile structurii fundamentale *iteraţie*

Primii algoritmi cu aplicabilitate în domeniul prelucrării automate a datelor se datorează, se pare, lui Charles Babbage, proiectantul „maşinii analitice" din anii 1830. O dată cu apariţia calculatoarelor electronice, programatorii s-au aplecat asupra algoritmilor ca baze ale conceperii programelor. Aceste preocupări au devenit tot mai intense pe măsură ce limbajele de programare s-au perfecţionat:

- *prima generaţie* (cod maşină) este etapa programării „artizanale" a calculatoarelor, în care programatorul putea fi uşor asemuit cu un „mare preot";
- *generaţia a doua* (limbaje de asamblare) simulează structurile de control prin instrucţiuni de salt (`jump`) şi apeluri de proceduri (`call`);
- *generaţia a treia* (limbaje de nivel înalt) dispune de instrucţiuni puternice pentru codificarea structurilor de control, pentru atribuire şi pentru calcul. Totodată apare programarea structurată, conform căreia o problemă mare este descompusă în module mai mici, ce pot fi atacate separat, urmând a fi recombinate;

- *generaţia a patra* adaugă la puterea limbajelor de generaţia a treia instrumente de asistare a programatorului capabile chiar să genereze cod-sursă conform unor specificaţii grafice şi textuale întocmite de programator.

Am folosit aici noţiunea de *cod-sursă*. Acesta este codul scris de om într-un anumit limbaj de programare, traducând instrucţiunile algoritmului în instrucţiuni ale respectivului limbaj. Trecerea de la codul-sursă la codul executabil are loc printr-un proces de interpretare (instrucţiunile în cod-sursă sunt traduse şi executate pe rând) sau de compilare (tot codul-sursă este mai întâi tradus în cod executabil, apoi pot avea loc oricâte execuţii ale programului fără a mai fi nevoie de „traducerea" prealabilă).

Codificarea structurilor de control în Visual FoxPro

Este de la sine înţeles că nu se poate concepe un program în programarea structurată fără a face apel la instrucţiunile care codifică structuri de control. Visual FoxPro implementează asemenea structuri prin câteva sintaxe specifice.

Structura alternativă (selecţia) se codifică prin:

```
IF <condiţie> [THEN]
   ...
[ELSE]
   [...]
ENDIF
```

sau, pentru structura alternativă generalizată:

```
DO CASE
CASE <conditie-1>
...
[CASE <conditie-2>]
[...]
[CASE <conditie-n>]
[...]
[OTHERWISE]
[...]
ENDCASE
```

Tot aici avem şi varianta funcţiei `IIF(<condiţie>,<valoare-dacă-adevărat>,<valoare-dacă-fals>)`, care se poate folosi atât în programe, cât mai cu seamă în scrierea regulilor de validare la nivel de câmp ori de înregistrare pe care le-am văzut în capitolul 2.

Structurile repetitive se codifică fie prin `DO WHILE...ENDDO`, fie prin `FOR... ENDFOR`:

```
DO WHILE <condiţie-de-ciclare>
...
[LOOP]
[EXIT]
ENDDO
```

Clauza `LOOP` trimite execuţia programului înapoi la linia `DO WHILE`, iar `EXIT` la următoarea instrucţiune după `ENDDO` (cu `LOOP` se poate relua, iar cu `EXIT` se poate părăsi bucla atunci când în interiorul acesteia se atinge o anume condiţie, alta decât cea de ciclare). Instrucţiunile din corpul buclei se execută atât timp cât condiţia precizată este adevărată.

Instrucțiunea FOR servește la codificarea structurii repetitive cu un număr finit de pași:

```
FOR VariabilăContor = ValoareInițială TO ValoareFinală
[STEP increment]
...
[EXIT]
[LOOP]
ENDFOR | NEXT
```

Clauzele LOOP și EXIT au aceleași roluri ca la instrucțiunea DO WHILE de mai sus.

Visual FoxPro, ca orice produs xBase, dispune și de o instrucțiune SCAN...ENDSCAN, care permite execuția unor anumite instrucțiuni concomitent cu parcurgerea secvențială a unei tabele .DBF:

```
SCAN [<domeniu> FOR <condiție-1> WHILE <condiție-2>]
...
[LOOP]
[EXIT]
ENDSCAN
```

Toate acestea vor fi exemplificate prin cod-sursă spre finalul acestui capitol.

4.3. Variabile și constante. Instrucțiuni de atribuire în Visual FoxPro

Fiecare limbaj are propriile sale instrucțiuni și propriile tipuri de date. Ați făcut cunoștință cu tipurile de date din Visual FoxPro în capitolul 2, iar cu câteva dintre comenzile modului de lucru direct v-ați întâlnit și în capitolul 3.

Atât variabilele, cât și constantele sunt abstractizări ale unor zone de memorie. Programatorii nu pot memora adrese și deplasamente de memorie, afară de cazul în care chiar le-ar plăcea acest exercițiu mental. În plus, este puțin predictibil ca o anumită variabilă să se găsească de două ori la aceeași adresă (compilatorul execută o alocare dinamică a adreselor, în funcție de memoria liberă la lansarea programului). Sunt două motive care cer folosirea de nume pentru aceste zone. Constantele nu au nevoie de nume; programatorul scrie expresiile lor direct în cod, ca de exemplu (sperăm că zecimalele lui π sunt suficiente):

Listing 4.1. Variabile și constante

```
STORE 0 to arie
STORE 20 to raza
ARIE=3.141592653*raza^2
?arie
```

Și constantele necesită spațiu în memorie, dar rezervarea acestuia nu cade în sarcina programatorului. Uneori însă este util să se dea și constantelor niște nume, pentru a le face mai ușor de manipulat; acest lucru este posibil în Visual FoxPro prin *directiva de precompilare* (un tip special de instrucțiune) #DEFINE:

Listing 4.2. Directiva #DEFINE

```
#DEFINE pi 3.141592653
STORE 0 to arie
STORE 20 to raza
arie=pi*raza^2
?arie
```


În treacăt fie zis, definiția de mai sus nu-și are rostul, deoarece Visual FoxPro dispune de funcția PI(), care are exact rolul de a returna numărul π cu primele 6 zecimale. Dar definiții de genul:

Listing 4.3. Directiva #DEFINE varianta 2

```
#DEFINE LINIENOUA CHR(13)+CHR(10)
#DEFINE MESAJ "Acest program a executat o operatiune incorecta " + ;
    "si va fi oprit de Windows." + linienoua +"Consultati asistenta tehnica."
#DEFINE TITLU "Eroare"
#DEFINE BUTOANE 1+16 && butoanele OK si Cancel, semnul STOP
MESSAGEBOX(MESAJ, BUTOANE, TITLU)
```

se dovedesc foarte utile, mai ales dacă trebuie invocată funcția MESSAGEBOX() în același context de mai multe ori pe parcursul execuției unui program. Este bine ca pentru numele de constante să folosim majuscule, iar pentru numele de variabile, litere mici.

Notă. În secvența de program de mai sus s-au folosit semnele &&, care preced un *comentariu*. Un alt caracter pentru introducerea comentariilor este asteriscul (*), plasat la începutul rândului. În loc de asterisc se poate folosi și cuvântul-cheie NOTE. Comentariile au rolul de a mări lizibilitatea programului.

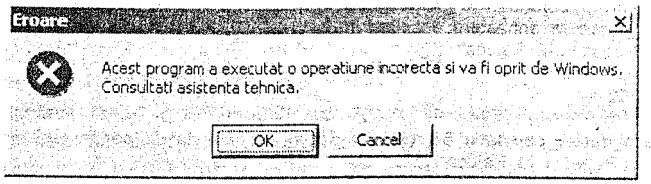


Figura 4.3. Rezultatul execuției funcției MESSAGEBOX() din codul de mai sus

Renunțarea la denumirea constantei se face cu directiva #UNDEF nume-constanta.

Lucrul cu variabile se dovedește a fi ceva mai complicat decât lucrul cu constante. În primul rând, pentru a folosi variabile, trebuie să le declarăm. Declararea implică rezervarea unui spațiu de memorie suficient de mare pentru respectivul tip de dată. În capitolul 2 am văzut care sunt tipurile de date folosite în bazele de date Visual FoxPro. Programele Visual FoxPro însă nu cer nici declararea explicită, nici specificarea unui tip pentru o variabilă (se spune că este un limbaj *weak typed* – în opoziție cu limbaje *strong typed*, care cer declararea explicită a variabilelor de un anumit tip: Pascal, C).

Începând cu tehnica programării structurate, se obișnuiește ca programul să fie compus din proceduri și funcții care se apelează unele pe altele. Limbajul de programare Visual FoxPro respectă și el paradigma procedurală. Din acest punct de vedere, se poate lucra cu variabile *publice* (după declarare, pot fi apelate în orice procedură sau funcție din cuprinsul aplicației) ori cu variabile *private* (sunt accesibile numai în interiorul procedurii sau funcției în care au fost declarate). În Visual FoxPro, pentru *declararea explicită* a variabilelor se pot folosi instrucțiunile:

- PUBLIC nume-variabila1[,nume-variabila2...]

pentru declararea unor variabile publice (globale) şi:

- `LOCAL nume-variabila1[,nume-variabila2...]`

în scopul declarării unor variabile locale. Instrucţiunea:

- `DIMENSION nume-tablou1(nrlinii1 [,nrcoloane1] [, nume-tablou2(nrlinii2 [,nrcoloane2]...]`

se foloseşte la declararea de tablouri cu o dimensiune (vectori) sau cu două dimensiuni (matrice), al căror domeniu de vizibilitate este tot local. Dacă se doreşte declararea unor tablouri globale, `DIMENSION` se înlocuieşte cu `PUBLIC ARRAY`.

Notă. Numele variabilelor pot conţine litere, cifre şi caracterul de subliniere (*underscore*) şi se recomandă să nu înceapă cu cifră, şi nici cu liniuţa de subliniere (pentru a evita confuzia cu *variabilele de mediu*, care au fost amintite în capitolul 1).

În Visual FoxPro există şi instrucţiunea `PRIVATE`, dar funcţiunea sa e puţin diferită faţă de alte limbaje precum `BASIC` sau `C`. Astfel, scrisă într-un subprogram (o procedură, după terminologia Visual FoxPro), declaraţia `PRIVATE <lista-de-variabile>` sau `PRIVATE ALL LIKE <şablon>`[20] permite „ascunderea" valorilor variabilelor globale specificate în listă faţă de programul curent (deci posibilitatea manipulării după dorinţă a unor variabile globale pe parcursul subprogramului, la ieşirea din acesta regăsind valorile lor originale):

Listing 4.4. Declararea variabilelor `PRIVATE`

```
CLEAR && sterge ecranul
PUBLIC m_nume, m_annastere, ancalend
m_nume="Popescu M. Raluca"
m_annastere=1979
ancalend=2001

DO afiseaza_datele_persoanei && apelul procedurii afiseaza_datele_persoanei
* se va afisa Popescu M. Raluca  1979  2001
?m_nume, m_annastere, ancalend
RELEASE ALL && sterge toate variabilele

PROCEDURE afiseaza_datele_persoanei && inceput de subprogram
PRIVATE ALL LIKE m_*   && se ascund variabilele publice
&& ale caror nume incep cu "m_"
m_nume="Ionescu Eugen"
m_annastere=1909
* se va afisa Ionescu Eugen  1909  2001
?m_nume, m_annastere, ancalend
RETURN && sfarsit de subprogram
```

Tipul unei variabile declarate explicit este stabilit o dată cu prima instrucţiune de atribuire care se referă la variabila respectivă şi, în linii mari, putem vorbi despre variabile de tip numeric, şir de caractere, dată calendaristică, logic şi obiect. De remarcat că variabilele publice (folosirea comenzii `PUBLIC` într-un program) au după declarare tipul de

20. Prin „şablon" (termenul folosit de Visual FoxPro este *skeleton*, literal „schelet") se înţelege orice combinaţie de litere, cifre, caracterul „_" şi caracterele „?" (înlocuieşte o literă ori o cifră) sau „*" (înlocuieşte un grup de caractere).

dată `Logical` şi valoarea `.F.`. La următoarele atribuiri, tipul de dată poate fi schimbat (cu alte cuvinte, o variabilă se poate refolosi).

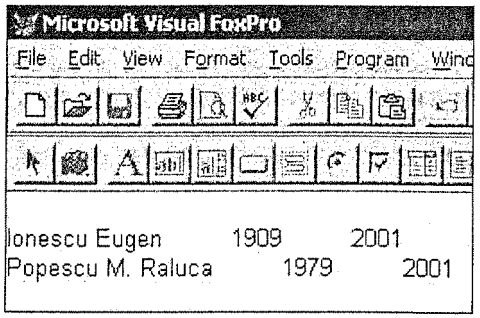


Figura 4.4. Rezultatul execuţiei programului din listingul 4.4

De asemenea, Visual FoxPro permite declararea implicită a variabilelor, adică o dată cu prima instrucţiune de atribuire. Instrucţiunile de atribuire sunt următoarele:

- operatorul „egal";

```
nume-variabila=valoare
```

- instrucţiunea `STORE`:

```
STORE valoare1 TO nume-variabila1
[,valoare2 TO nume-variabila2...]
```

Pentru variabilele de tip obiect se foloseşte operatorul „=", combinat cu funcţia `CREATEOBJECT(<nume-bibliotecă.nume-clasă>)`, pe care o vom mai întâlni în capitolul 11.

Notă. Variabilele declarate implicit sunt locale, în afara celor declarate ad-hoc în fereastra de comenzi, care sunt globale.

Ulterior creării unei variabile, tipul acesteia se poate verifica prin invocarea funcţiei `TYPE (<nume-variabilă>)`. Aceasta returnează un număr care arată tipul variabilei (vezi tabelul 4.1).

Tabelul 4.1. Valori posibile ale funcţiei `TYPE`

Rezultat	*Tip de dată*
C	Character
N	Numeric (inclusiv float, double, integer)
Y	Currency
D	Date
T	DateTime
L	Logical
M	Memo
O	Object
G	General
U	Nedefinit (variabila nu există)

Notă. Oricare ar fi tipul variabilei, numele său se scrie între ghilimele sau apostrofuri (în Visual FoxPro aceste semne au acelaşi sens).

Funcția VARTYPE(nume-variabila,[valoare pentru NULL]) are același rol ca și TYPE(), dar este mai rapidă și nu necesită scrierea numelui variabilei între ghilimele; în plus, cel de-al doilea argument, opțional, permite specificarea unui rezultat convențional în caz că variabila este NULL. Astfel, dacă al doilea argument are valoarea .T. și variabila este NULL, se va returna tipul variabilei; dacă al doilea argument este omis sau este .F., se va returna „X", ceea ce înseamnă NULL. Presupunând că var1 este de tip N (numeric) și deocamdată are valoarea NULL, VARTYPE(var1, .T.) va returna rezultatul „N", pe când VARTYPE (var1) ori VARTYPE (var1,.F.) vor returna ambele rezultatul „X".

Visual FoxPro permite și folosirea variabilelor de tip tablou (cu o dimensiune – vector sau cu două dimensiuni – matrice). Acestea se declară cu DIMENSION sau cu PUBLIC. Pentru adresarea elementelor se folosește notația <nume-tablou>(i,j). De asemenea, pentru tablouri sunt disponibile multe funcții utile cum ar fi:

ALEN(nume-tablou[, atribut]) – pentru aflarea numărului total de elemente (fără argumentul atribut), a numărului de linii (dacă atribut este 1) ori de coloane (dacă atribut este 2);

ASCAN(nume-tablou, expresie-căutată [, element-de-plecare [, număr-elemente-de-parcurs]]) – află numărul de ordine al elementului care conține expresie-căutată; returnează 0 (zero) dacă nu există un astfel de element.

Orice variabilă Visual FoxPro poate fi distrusă explicit prin una dintre comenzile:

CLEAR ALL – șterge necondiționat din memorie toate variabilele; comanda CLEAR are însă un spectru mai larg;

RELEASE <lista-de-variabile> – șterge din memorie variabilele specificate;

RELEASE ALL – șterge din memorie orice variabilă ori tablou;

RELEASE ALL LIKE <șablon>| EXCEPT <șablon> – șterge variabilele care sunt/nu sunt conforme cu șablonul.

Notă. Dacă este integrată într-un program, comanda RELEASE ALL nu afectează variabilele publice. În caz că se dorește acest lucru, se va folosi comanda RELEASE ALL EXTENDED.

Visual FoxPro permite și păstrarea valorilor unor variabile în fișiere speciale, denumite fișiere memo. Astfel, puternica instrucțiune:

SAVE TO <nume-fisier> ALL LIKE <sablon> |ALL EXCEPT <șablon>

memorează într-un fișier variabilele care sunt/nu sunt conforme șablonului, iar comanda:

RESTORE FROM <nume-fisier> [ADDITIVE]

reface variabilele stocate anterior în fișierul specificat; clauza ADDITIVE împiedică distrugerea variabilelor care deja există în memorie. De remarcat că variabilele își păstrează după restaurare caracterul public sau local.

Comenzile SAVE TO... și RESTORE FROM... își găsesc utilitatea în stabilirea unor opțiuni de configurare a unei aplicații (parole, directoare de lucru, luni calendaristice etc.).

Faptul că Visual FoxPro nu impune declararea explicită a variabilelor și nici o consecvență privind tipurile de date pe parcursul execuției programului oferă o mare flexibilitate, dar poate constitui și o sursă de erori.

4.4. Instrucțiuni de intrare-ieșire

În Visual FoxPro, cea mai simplă instrucțiune de ieșire este fără îndoială „?". Ea are ca scop afișarea pe ecranul principal a unei expresii (constantă, variabilă, combinații de constante și variabile etc.). Fiecare comandă ? începe afișarea din coloana 0 (marginea stângă) și produce un avans de rând. Dacă se folosesc două semne de întrebare în loc de unul, afișarea are loc pe linia curentă, iar dacă se folosesc trei semne de întrebare, afișarea va avea loc direct la imprimantă.

O altă instrucțiune de ieșire mai veche este instrucțiunea de afișare formatată:

@ <linie>, <coloana> SAY <expresie>

Ambele instrucțiuni prezentate se utilizau pe larg în versiunile FoxPro pentru DOS, în care afișarea pe ecran avea loc în mod caracter (25 de linii și 80 de coloane). Cât despre afișarea la imprimantă a unor situații complexe, aceasta a avut întotdeauna la bază machetele de raport, cu care ne vom întâlni în capitolul 7.

Ca instrucțiuni de intrare, Visual FoxPro, menține mai vechile INPUT și ACCEPT, ca și:

@ <linie>, <coloana> GET <variabilă>, urmată de READ.

În versiunile Visual FoxPro, aceste instrucțiuni sunt considerate depășite. Rolul instrucțiunilor de intrare/ieșire (pentru afișarea pe ecran) a fost preluat de obiectele de control (Label, TextBox, ListBox etc.) utilizate în formulare, în care modul de afișare este exclusiv grafic (vezi capitolul 8). Instrucțiunile @...SAY...GET...READ, INPUT, ACCEPT sunt menținute din rațiuni de compatibilitate cu aplicațiile concepute în mod text. Instrucțiunea „?" se poate folosi în etapa testării și depanării unor programe, întrucât oferă o modalitate simplă și rapidă de afișare.

4.5. Proceduri și funcții în Visual FoxPro

Programarea structurată ne învață că o aplicație este cu atât mai simplu de întreținut cu cât codul-sursă este mai bine împărțit în module cât mai mici și specializate.

Modularea codului constituie o metodologie de construire a unui program pe baza unor părți distincte (module), care îndeplinesc fiecare o sarcină specifică în sensul atingerii scopului final al programului. Această tehnică (ca și programarea structurată în sine) își are rădăcinile în practica de toate zilele, ea prezentând o serie de avantaje majore și anume:

- flexibilitatea și posibilitatea de reutilizare a codului, acestea fiind cu atât mai puternice cu cât modulele sunt mai mici și mai specializate;
- lizibilitatea și ușoara mentenanță a unei aplicații în totalitatea ei.

Diferența între proceduri și funcții trebuie privită mai mult din punct de vedere teoretic-conceptual, ea evidențiindu-se chiar din definițiile celor două tipuri de module.

O *procedură* reprezintă o înșiruire logică a unui număr finit de instrucțiuni și structuri de control grupate sub același nume și care duc la îndeplinire o sarcină specifică; poate fi apelată de alte module – prin comanda DO <nume-procedură> (eventual cu unul sau mai mulți parametri), dar fără a returna neapărat vreo valoare (figura 4.5).

Funcția are o definiție aproape identică, numai că, obligatoriu, va returna o valoare (una și numai una).

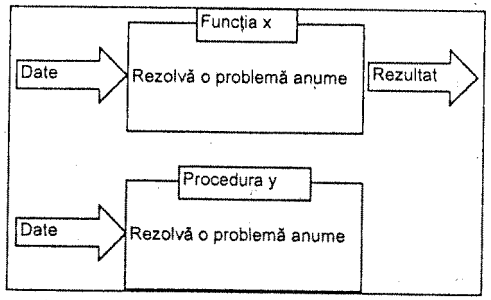


Figura 4.5. Diferența dintre proceduri și funcții

Așadar, spre deosebire de proceduri care pot fi apelate direct (ele funcționând de sine stătător), o funcție are întotdeauna un rezultat și, ca urmare, o putem utiliza prin asignarea acestui rezultat unei variabile sau prin includerea ei într-o comandă Visual FoxPro sau într-o frază SELECT - SQL. Există și varianta =numefuncție() (fără nici o variabilă în stânga egalului).

Crearea programelor (proceduri sau funcții) în Visual FoxPro presupune parcurgerea următorilor pași:

- lansarea editorului de text propriu prin utilizarea în fereastra de comandă a comenzii: MODIFY COMMAND [<NumeProgram>] ce va deschide o fereastră specifică pentru introducerea secvențelor de cod dorite.
- altă posibilitate o constituie utilizarea Project Manager-ului, cadrul de pagină Code și utilizarea butoanelor New sau Modify, după necesități. Aceasta este, de altfel, modalitatea standard de lucru pentru crearea programelor, ea fiind recomandată datorită faptului că utilizarea comenzii de mai sus duce la crearea programului, dar nu și la includerea lui în proiect (aspect foarte important atunci când dorim livrarea unor aplicații compilate în format executabil).
- compilarea codului-sursă (opțiunea Program|Compile din meniu), operațiune care va duce la transformarea lui într-un format interpretabil de UCP. Compilatorul va asigura și corectitudinea sintaxei, astfel încât orice inadvertență în scrierea unei instrucțiuni sau utilizarea defectuoasă a structurilor de control va genera un mesaj de eroare specific. Compilarea are loc și dacă în fereastra de comenzi se scrie COMPILE <nume-program>|<șablon[21]>;
- lansarea codului în execuție – se realizează fie din Project Manager, prin selecția programului dorit în cadrul de pagină Code și click pe butonul Run, fie prin utilizarea (în fereastra de comandă sau într-o procedură apelantă) a comenzii: DO NumeProgram [WITH <lista-parametri>].

Notă. Întreruperea execuției unui program se realizează apăsând tasta Esc.

Codul-sursă Visual FoxPro se memorează în fișiere cu extensia .PRG. Codul compilat se găsește în fișiere cu extensia .FXP (dacă nu a avut loc compilarea explicită, un asemenea fișier se creează automat la lansarea în execuție a programului .PRG corespunzător). Putem avea însă cod-sursă Visual FoxPro și în alte tipuri de fișiere: procedurile stocate și

21. De exemplu, compile m*.prg va compila toate programele ale căror nume încep cu litera *m*.

declanșatoarele (fac parte din dicționarul de date reprezentat de cuplul de fișiere .DBC/.DCT), meniurile (fișierele generate, care au extensia .MPR) și formularele (fișierele .SCT).

Orice funcție sau procedură poate fi declarată (construită) în 3 moduri:

- ca program distinct, caz în care va fi accesibilă (va putea fi apelată) din oricare alt modul (ca în exemplele anterioare);
- ca parte integrantă a unei alte proceduri sau funcții, caz în care va fi accesibilă în mod normal numai modulului respectiv, fiind „invizibilă" altor programe[22]. Asemenea module se declară *întotdeauna la sfârșitul* secvenței de cod ce constituie programul apelant (pentru exemplificare, vezi și capitolul 7, partea referitoare la rapoarte dinamice) astfel:
- pentru proceduri:

```
PROCEDURE nume_proc
[PARAMETERS parametri...]
      <instrucțiuni>
ENDPROC
```

- pentru funcții:

```
FUNCTION nume_funct
[PARAMETERS parametri...]
      <instrucțiuni>
RETURN valoare
ENDFUNC
```

Notă. RETURN fără valoare realizează ieșirea forțată din modul și reîntoarcerea la instrucțiunea imediat următoare din modulul apelant.

O funcție sau o procedură poate fi creată și ca procedură stocată (capitolul 6), caz în care va fi integrată în dicționarul bazei de date, fiind disponibilă oricărui modul de program, cu condiția ca respectiva bază de date să fie deschisă.

Atunci când într-un program (sau chiar în fereastra de comenzi) este apelată o procedură sau o funcție definită de utilizator, Visual FoxPro caută modulul respectiv în următoarele locuri (în ordine):

- programul curent (cel în care se află apelul funcției sau procedurii);
- programul specificat într-o comandă DO <nume-procedura> IN <nume-program> sau dacă înainte de instrucțiunea de apel (DO <nume-procedura>) s-a introdus comanda SET PROCEDURE TO <nume-program>;
- procedurile stocate;
- toate modulele de program din directorul curent și din directoarele specificate în comanda SET PATH.

Figura 4.6 prezintă toate elementele necesare lucrului cu programe în Visual FoxPro.

22. Vezi, în continuare, modalitatea de apel cu DO <nume-procedură> IN <nume-program> precum și comanda SET PROCEDURE TO.

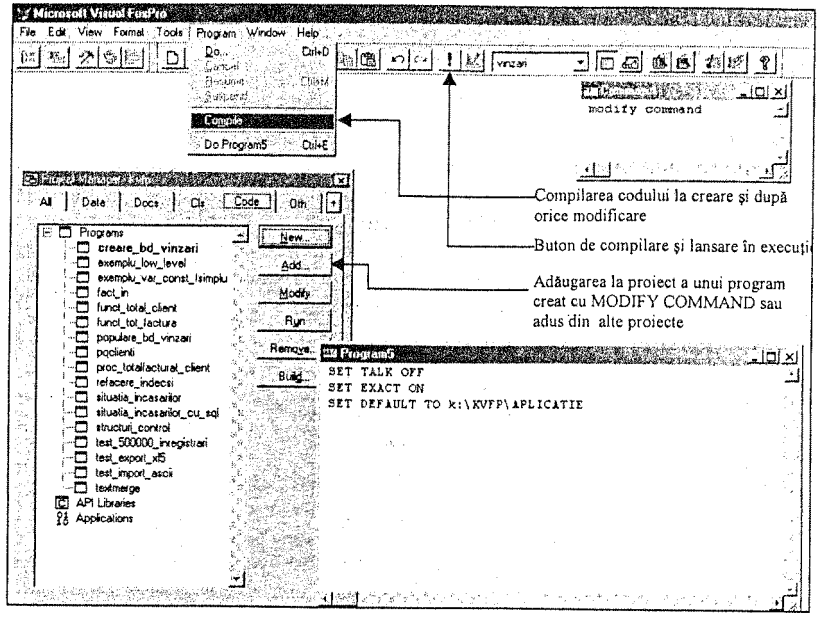


Figura 4.6. Lucrul cu programe în Visual FoxPro

După cum spuneam, o procedură sau o funcţie poate fi apelată în două moduri: cu parametri (unul sau mai mulţi, în funcţie de specificul modulului respectiv) sau fără parametri. Parametrii sunt variabile locale (declarate obligatoriu explicit la începutul modulului[23] apelat) care preiau valorile furnizate de modulul apelant în scopul utilizării lor în prelucrările interne. Putem vorbi deci despre:

- parametri formali – numele variabilelor locale declarate la începutul procedurii sau funcţiei;
- parametri actuali – valorile efective preluate de la modulul apelant.

Aşadar, în Visual FoxPro, obligatoriu prima instrucţiune a unei proceduri se referă la declararea parametrilor formali cu ajutorul sintaxei:

```
PARAMETERS ParamFormal1, ParamFormal2 ...
```

În loc de declaraţia PARAMETERS se poate utiliza şi LPARAMETERS.

Apelarea unei proceduri cu parametri se realizează conform sintaxei:

```
DO NumeProcedură WITH ParamActual1, ParamActual2...
```

Apelarea unei funcţii diferă radical din punct de vedere al sintaxei:

```
Variabilă=NumeFuncţie(ParamActual1, ParamActual2...)
```

Sau (apel direct într-o frază SQL – cu care ne vom întâlni în capitolul 5):

```
SELECT câmp1,câmp2,...,NumeFuncţie(ParamActual1,
       ParamActual2...) [AS alias ]
```

Ordinea în care sunt preluaţi parametrii actuali de către cei formali este cea naturală, de la stânga la dreapta, în funcţie de virgulă ca separator, iar tipul parametrilor formali este

23. Prin modul se înţelege o procedură sau o funcţie.

determinat de tipul parametrilor actuali. Dacă apelul se face fără nici un parametru actual, cei formali sunt declaraţi implicit de tip logic (boolean).

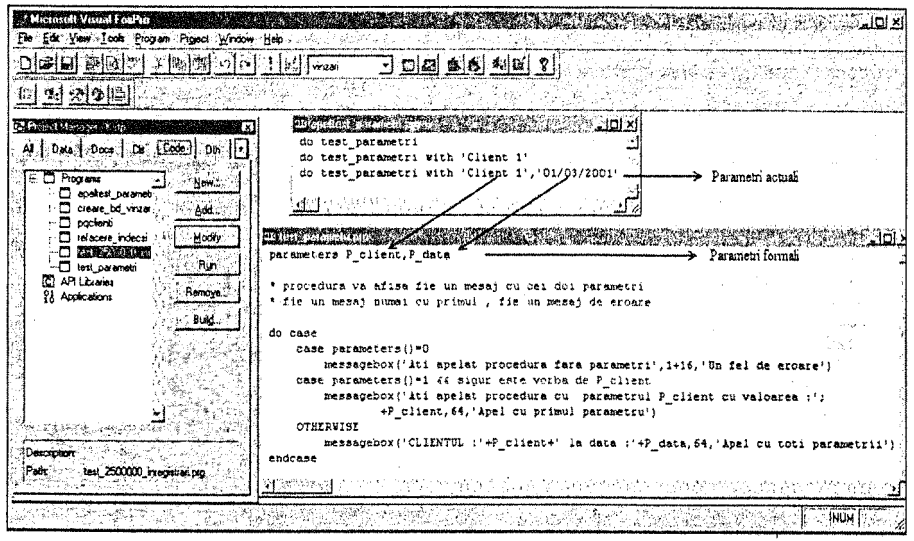


Figura 4.7. Apelul procedurilor şi transmiterea parametrilor în Visual FoxPro

Figura 4.7 prezintă o procedură cu numele `Test_Parametri` şi o procedură cu numele `ApelTest_Parametri` care o va apela pe prima cu şi apoi fără parametri, tocmai în ideea de a demonstra modalitatea standard de lucru cu proceduri în Visual FoxPro. Funcţia `PARAMETERS()` returnează numărul de parametri actuali cu care este apelat un modul. *Un parametru formal care nu are la un moment dat un corespondent actual este întotdeauna de tip logic şi are valoarea* `.F. (False)`.

În Visual FoxPro, de la un modul apelant către un modul apelat se pot transmite maximum 27 de parametri. Modul implicit de transmitere în Visual FoxPro este *transmiterea prin valoare* (modulul apelat lucrează cu copii ale variabilelor transmise de modulul apelant). Pentru a trece la *transmiterea prin referinţă* (modulul apelat preia adresele variabilelor transmise de modulul apelant, deci ambele module lucrează cu exact acelaşi set de date), se poate folosi comanda de configurare `SET UDFPARMS TO REFERENCE`, care are ca efect transmiterea tuturor parametrilor prin referinţă. Dacă se doreşte transmiterea numai anumitor parametri prin referinţă, atunci parametrul dorit se prefixează cu `@`. Astfel se pot transmite ca parametri tablouri (*arrays*) întregi (transmiterea prin valoare ar considera numai primul element al tabloului). O altă utilitate a transmiterii prin referinţă apare atunci când se apelează programe/proceduri externe (scrise în C sau C++, de pildă) care cer ca parametri referinţe ori pointeri.

Listing 4.5. Ilustrarea transmiterii parametrilor prin referință

```
* declararea unui tablou pentru pastrarea datelor salariale
DIMENSION vSp2001(3,4)
*-----------------------------------------
* consideram urmatoarea structura a sporurilor
*       1          2          3              4
*    salar       vechime  cond_grele       altele
*-----------------------------------------
* popularea tabloului cu date de test pentru 3 salariati
* prima linie:
vSp2001(1,1)=2456000
vSp2001(1,2)=0.2
vSp2001(1,3)=0.15
vSp2001(1,4)=365000
* a doua linie:
vSp2001(2,1)=3500000
vSp2001(2,2)=0.15
vSp2001(2,3)=0.0
vSp2001(2,4)=100000
* a treia linie:
vSp2001(3,1)=4200000
vSp2001(3,2)=0.3
vSp2001(3,3)=0.02
vSp2001(3,4)=0

* apelul procedurii: tabloul se transmite prin adresa
CalcSporuri(@vSp2001)


* procedura pentru calculul si afisarea sporurilor
PROCEDURE CalcSporuri
LPARAMETERS vSporuri
local lnSpor
FOR i = 1 TO ALEN(vSporuri,1) && pentru fiecare linie din tablou
lnSpor=vSporuri(i,1)*(vSporuri(i,2)+vSporuri(i,3))+vSporuri(i,4)
? "Salariatul nr. " + allt (str(i)) + ;
    " are salariul de baza de " +allt(str(vSporuri(i,1))) +;
    " si totalul sporurilor de " + allt(str(lnSpor))
ENDFOR
ENDPROC
```

Într-o primă etapă, apelul procedurii (linia redată în caracter **bold**) fusese scrisă astfel:

`CalcSporuri(vSp2001)` – deci transmiterea valorii, care este implicită în Visual FoxPro. Acest lucru cauzează o eroare, vizibilă în figura 4.8.


```
* apelul procedurii
CalcSporuri(vSp2001)


* procedura pentru calculul si afisarea sporurilor
PROCEDURE CalcSporuri
LPARAMETERS vSporuri
local lnSpor
FOR i = 1 TO ALEN(vSporuri,1) && pentru fiecare linie din tablou
lnSpor=vSporuri(i,1)*(vSporuri(i,2)+vSporuri(i,3))+vSporuri(i,4)
? "Salariatul nr. " +
    " are salariul de
    " si totalul sporu
ENDPROC
```

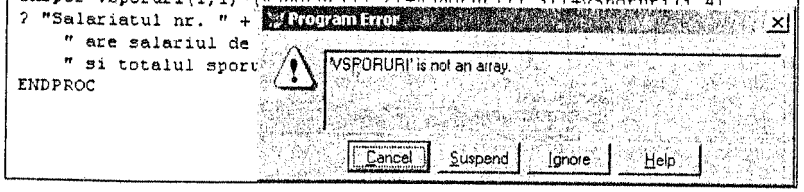


Figura 4.8. Transmiterea valorii unui tablou în procedura apelată cauzează o eroare

După corectarea erorii, rezultatul execuției este cel din figura 4.9.

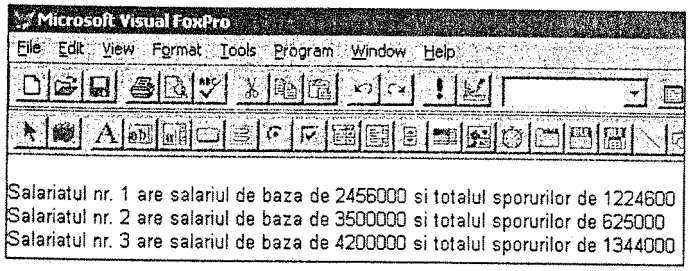


Figura 4.9. Rezultatele programului din listingul 4.5

Vom prezenta în continuare câteva exemple de programe în Visual FoxPro.

Exemplul 1

Dată fiind baza de date `VINZARI`, dorim să aflăm care sunt facturile încasate integral. În acest scop avem nevoie de datele din tabelele FACTURI şi INCASFACT. Vom parcurge tabela FACTURI, căutând pentru fiecare număr de factură înregistrările corespunzătoare (dacă sunt) în tabela INCASFACT şi cumulând valorile câmpului `Transa`.

Pentru înțelegerea deplină a modului de calcul, algoritmul a fost redat sub forma schemelor logice din figurile 4.10 şi 4.11.

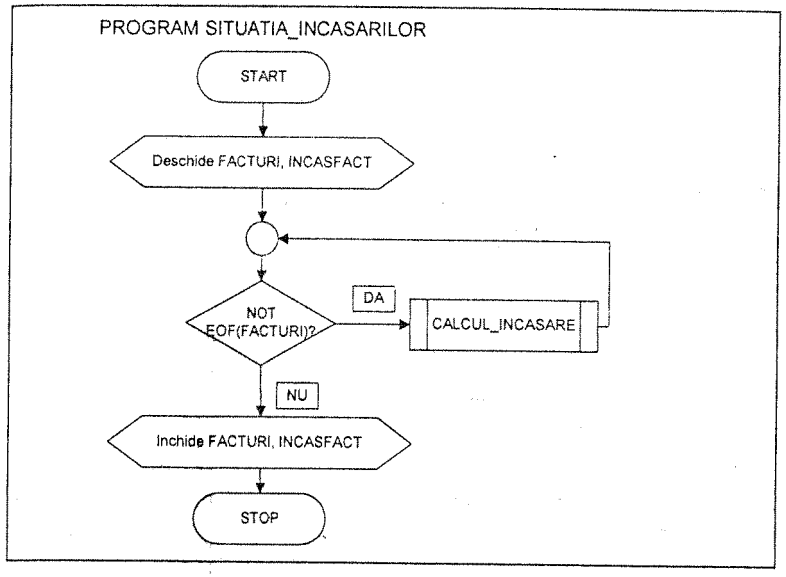


Figura 4.10. Schema logică pentru programul principal SITUATIA_INCASARILOR

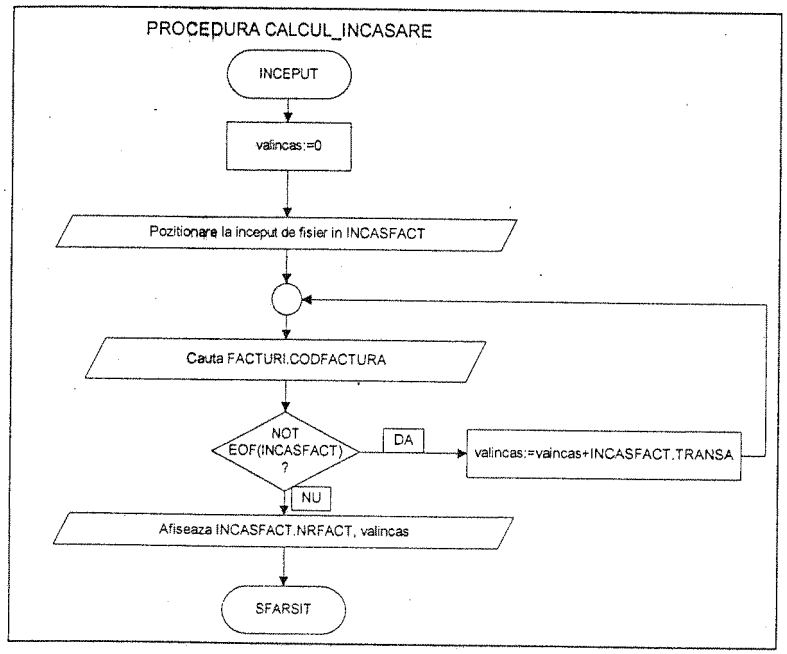


Figura 4.11. Schema logică pentru procedura CALCUL_INCASARE

Codul-sursă al programului Visual FoxPro SITUATIA_INCASARILOR.PRG este prezentat în listingul 4.6.

Listing 4.6. Varianta cu DO WHILE

```
* program pentru afisarea situatiei incasarilor
*********************************************
CLEAR
SET TALK OFF && pentru inhibarea afisarii mesajelor-sistem
STORE 0 TO nrf, valfact, valincas
* deschidem baza de date
IF !DBUSED("vinzari")
   OPEN DATABASE "k:\database\vinzari" SHARED
ENDIF
SET DATABASE TO vinzari && BD vinzari devine curenta
* deschidem tabelele
IF !USED("facturi")
   USE facturi IN 0 SHARED
ENDIF
IF !USED("incasfact")
   USE incasfact IN 0 SHARED
ENDIF
SELECT facturi
GO TOP
? "factura nr. ", "valoare cu TVA", "valoare incasata", "valoare de incasat"
DO WHILE NOT EOF()
   nrf=facturi.nrfact
   valfact=facturi.valftva*1.19
   SELECT incasfact
   SUM TRANSA TO valincas FOR nrfact=nrf
   ? nrf, valfact, valincas, valfact-valincas
   SELECT facturi
   SKIP
ENDDO
SET TALK ON
CLOSE TABLES ALL
CLOSE DATABASE
*********************************************
```

BIBL. CENTR. UNIV. „Al. I. CUZA" IAȘI INFORMATICĂ

În locul combinației DO WHILE NOT EOF() și SKIP se poate utiliza structura SCAN...ENDSCAN, care are avantajul că nu necesită o poziționare prealabilă la începutul tabelei și realizează un avans automat la înregistrarea următoare. Partea din codul-sursă scrisă în caractere **bold** se poate modifica după cum se arată în listingul 4.7.

Listing 4.7. Varianta cu SCAN și SUM

```
SELECT facturi
? "factura nr. ", "valoare cu TVA", "valoare incasata", "valoare de incasat"
SCAN
   nrf=facturi.nrfact
   valfact=facturi.valftva*1.19
   SELECT incasfact
   SUM TRANSA TO valincas FOR nrfact=nrf
   ? nrf, valfact, valincas, valfact-valincas
   SELECT facturi
ENDSCAN
```

În plus, comanda SCAN permite chiar parcurgerea condiţională a tabelei (SCAN NEXT 3 înseamnă în cazul de mai sus consultarea situaţiei primelor trei facturi). În acest program am beneficiat şi de pe urma comenzii SUM, fără de care programul ar fi fost mai complicat şi mai greu lizibil, din cauza prezenţei unei structuri repetitive suplimentare. Redăm în listingul 4.8, cu caractere **bold**, porţiunea modificată.

Listing 4.8. Varianta fără comanda SUM

```
SELECT facturi
? "factura nr. ", "valoare cu TVA", "valoare incasata", "valoare de incasat"
SCAN
    nrf=facturi.nrfact
    valfact=facturi.valftva*1.19
    valincas=0
    SELECT incasfact
    GO TOP
    DO WHILE NOT EOF()
        LOCATE FOR nrfact=nrf
        IF FOUND()
                valincas=valincas+transa
                DO WHILE NOT EOF()
                        CONTINUE
                        valincas=valincas+transa
                ENDDO
        ENDIF
    ENDDO
    ? nrf, valfact, valincas, valfact-valincas
    SELECT facturi
ENDSCAN
```

În interpretarea acestui program trebuie să ţineţi cont şi de faptul că, dacă instrucţiunea LOCATE/CONTINUE nu găseşte nici o înregistrare care îndeplineşte condiţia specificată, atunci funcţia EOF() are valoarea .T.

Variantele de program prezentate mai sus nu prea respectă schema logică, în care calculul valorii încasate pentru fiecare factură se realiza printr-o funcţie distinctă. O variantă modulară a programului este oferită în listingul 4.9.

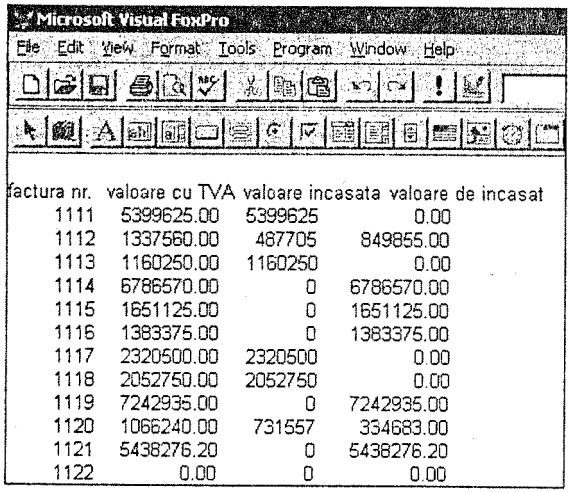
Listing 4.9. Variantă care foloseşte o procedură şi funcţie definită de utilizator

```
* program pentru afisarea situatiei incasarilor – varianta structurată
**********************************************
CLEAR
SET TALK OFF && pentru inhibarea afisarii mesajelor-sistem
STORE 0 TO nrf, valfact, valincas
* deschidem baza de date
IF !DBUSED("vinzari")
    OPEN DATABASE "k:\database\vinzari" SHARED
ENDIF
SET DATABASE TO vinzari && BD vinzari devine curenta
* deschidem tabelele
IF !USED("facturi")
    USE facturi IN 0 SHARED
ENDIF
IF !USED("incasfact")
    USE incasfact IN 0 SHARED
ENDIF
```



```
SELECT facturi
? "factura nr. ", "valoare cu TVA", "valoare incasata", "valoare de incasat"
SCAN
    DO calculeaza_si_afiseaza
ENDSCAN
SET TALK ON
CLOSE TABLES ALL
CLOSE DATABASE
**********************************************

PROCEDURE calculeaza_si_afiseaza
    nrf=facturi.nrfact
    valfact=facturi.valftva*1.19
    SELECT incasfact
    valincas=insumeaza_valoare(nrf)
    ? nrf, valfact, valincas, valfact-valincas
    SELECT facturi
ENDPROC

**********************************************
FUNCTION insumeaza_valoare
PARAMETERS nr
    STORE 0 TO vi
    SUM TRANSA TO vi FOR nrfact=nr
RETURN vi
```

Oricare ar fi varianta utilizată, rezultatul execuţiei arată ca în figura 4.12.



factura nr.	valoare cu TVA	valoare incasata	valoare de incasat
1111	5399625.00	5399625	0.00
1112	1337560.00	487705	849855.00
1113	1160250.00	1160250	0.00
1114	6786570.00	0	6786570.00
1115	1651125.00	0	1651125.00
1116	1383375.00	0	1383375.00
1117	2320500.00	2320500	0.00
1118	2052750.00	2052750	0.00
1119	7242935.00	0	7242935.00
1120	1066240.00	731557	334683.00
1121	5438276.20	0	5438276.20
1122	0.00	0	0.00

Figura 4.12. Situaţia încasării facturilor

O variantă mult mai simplă (folosind instrucţiuni SQL) se obţine prin combinarea a două comenzi SELECT: prima cumulează tranşele încasate din fiecare factură, a doua efectuează o joncţiune externă între tabela FACTURI şi tabela temporară cu încasări cumulate.

Listing 4.7 bis. Varianta cu interogări SQL

```
*************************************************
CLEAR
* deschidem baza de date
IF !DBUSED("vinzari")
    OPEN DATABASE "k:\database\vinzari" SHARED
ENDIF
SET DATABASE TO vinzari && BD vinzari devine curenta

* selectam intr-o tabela temporara incasarile grupate pe facturi
SELECT nrfact, SUM(TRANSA) AS val_incasata ;
    INTO CURSOR temp1 ;
    FROM incasfact GROUP BY nrfact

* efectuam o jonctiune externa
* cu tabela FACTURI
* (sunt si facturi pentru care nu este nici o plata)
SELECT F.nrfact AS factura_nr, F.valftva*1.19 AS val_cu_tva, ;
    NVL(val_incasata,0) as val_incasata, ;
    NVL(F.valftva*1.19-val_incasata,0) AS val_de_incasat ;
    FROM facturi F LEFT OUTER JOIN temp1 T ON T.nrfact=F.nrfact

CLOSE TABLES ALL
CLOSE DATABASE
*************************************************
```

Ați observat în programul de mai sus utilizarea frecventă a caracterului „punct și virgulă" folosit în scopul fragmentării pe mai multe rânduri a unei comenzi lungi.

Exemplul 2

Pentru evidențierea modului de lucru cu funcții, precum și a câtorva dintre structurile de control enumerate anterior vom recurge la exemplul calculului totalului facturat pe fiecare client în parte, varianta „pură xBase" (fără utilizarea SQL). Pentru îndeplinirea acestui obiectiv vom parcurge următorii pași:

- construim o funcție (`Funct_tot_factura` – figura 4.15) care totalizează liniile unei facturi inclusiv TVA (procentul este luat din tabela PRODUSE pentru fiecare produs în parte), ce va avea ca parametru numărul de factură;
- construim o funcție (`Funct_total_client` – figura 4.14) care totalizează facturile fiecărui client primind ca parametru codul clientului și va apela funcția `Funct_tot_factura` pentru fiecare factură a clientului x (transmițând ca parametru numărul facturii);
- construim, în final, o procedură (`Proc_totalfacturat_client` – figura 4.13) care parcurge tabela CLIENTI și, pentru fiecare client, apelează funcția `Funct_total_client` transmițând ca parametru codul clientului curent și afișând apoi un mesaj cu totalul clientului respectiv.


```
* totaluri facturi pe clienti fara utilizarea SQL
if  not used('clienti')
    use clienti in 0
endif

select clienti
go top

do while not eof()
    Vtotal_client= funct_total_client(clienti.codcl)
    messagebox('Totalul facturat pentru clientul :'+clienti.dencl+'este : ' ;
            +str(Vtotal_client),64,'Mesaj')
    select clienti
    skip
enddo
```

Figura 4.13. Procedura de afișare a totalului facturat pentru un client

```
parameters Pcodcl_

if !used('facturi')
    use facturi in 0
endif
select facturi
    Vtotal_facturi=0
scan for codcl=Pcodcl_
    Vtotal_facturi=Vtotal_facturi+funct_tot_factura(facturi.nrfact)
    select facturi
endscan

return  Vtotal_facturi
```

Figura 4.14. Funcția de calcul al totalului facturat pentru un client

```
parameters Pnrfact_
if !used('liniifact')
    use liniifact in 0
endif
if !used('produse')
    use produse in 0
endif

select liniifact
Vtotal_factura=0

scan for nrfact=Pnrfact_
    total_fara_tva=cantitate*pretunit
    * pozitionarea pointer-ului pe linia corespunzatoare codului curent in Produse:
        cauta=seek(liniifact.codpr,'Produse','Codpr')
    tva_=produse.proctva*total_fara_tva
    Vtotal_factura=Vtotal_factura+total_fara_tva+tva_
    select liniifact
endscan
return Vtotal_factura
```

Figura 4.15. Funcția de calcul al totalului unei facturi

4.6. Macrosubstituția: într-un cuvânt, mai multă putere

Putem spune că macrosubstituția este „mană cerească" pentru orice dezvoltator de aplicații mai „răsărite" care se confruntă continuu cu problema generalizării modulelor. Unul dintre obiectivele urmărite la o aplicație poate fi: cât mai puțin cod-sursă care să răspundă unor necesități cât mai variate și în cât mai multe moduri posibil. Cu alte cuvinte, o aplicație informatică profesională trebuie să vină în întâmpinarea cerințelor noi ale utilizatorilor și chiar trebuie să le provoace pentru a-i mulțumi pe deplin. Programatorul nu trebuie întotdeauna să identifice și să codifice opțiuni statice pentru necesități specifice, ci, de multe ori, trebuie să aibă o viziune largă asupra spectrului de posibile opțiuni și cerințe și să construiască module cu un comportament dinamic, flexibile și în același timp fiabile (in limita posibilului).

Așadar, macrosubstituția dotează cu un comportament dinamic diverse obiecte ale unei aplicații. Ea se realizează prin utilizarea semnului & (*ampersand*) înaintea unei variabile de tip caracter. Această variabilă stochează o instrucțiune sau o expresie evaluabilă și se execută astfel:

```
&nume-variabilă
```

Spre exemplu, secvența următoare este introdusă în fereastra de comandă:

```
x='1+1'
? x
? &x
```

Ea va avea următorul rezultat (vezi și figura 4.16):

- se creează variabila de tip caracter „x" și i se atribuie șirul de caractere „1+1";
- în fereastra principală se va afișa (comanda ?) șirul ca atare: 1+1;
- în fereastra principală se afișează numărul 2 ca rezultat al evaluării expresiei „1+1".

De asemenea, secvențele:

```
x='liniifact'
use &x in 0
```

sau

```
x= 'use liniifact in 0'
&x
```

vor avea ca rezultat deschiderea tabelei LINIIFACT.

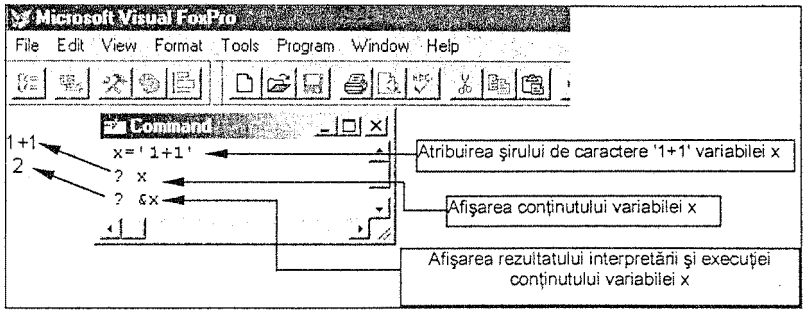


Figura 4.16. Exemplu de macrosubstituție

Accentuăm faptul că șirul de caractere atribuit variabilei ce va constitui sursa macrosubstituției trebuie să fie o expresie corectă din punct de vedere sintactic și interpretabilă de compilatorul Visual FoxPro. Șirul respectiv nu poate conține el însuși semnul &.

Următoarele listinguri sunt, sperăm, suficient de concludente pentru a demonstra utilitatea fantastică a acestui instrument numit macrosubstituție și pe care, din păcate, nu-l regăsim în multe SGBD-uri sau limbaje de programare.

Listing 4.10. Deschiderea tuturor tabelelor bazei de date curente

```
* exemplu de cod generalizat pentru
* deschiderea tabelelor unei baze de date
* varianta cu macrosubstitutie &
CLOSE TABLES ALL
clear
* extragem din containerul bazei de date(este tot o tabela) numele tabelelor
OPEN DATABASE "k:\aplicatie\database\vinzari" SHARED
*functia ADBOBJECTS populeaza un vector cu obiectele bazei de date curente
ADBOBJECTS(vTabele, "TABLE")
FOR i= 1 TO ALEN(vTabele,1)
    USE &vTabele(i) IN 0
ENDFOR
```

În listingul 4.10 s-a utilizat una din multele funcții „utilitare" ale Visual FoxPro, anume `ADBOBJECTS(nume-tablou, tip-obiect)`, care are rolul de a plasa într-un tablou numele obiectelor din baza de date deschisă. Filtrarea acestor obiecte depinde de valoarea celui de-al doilea argument al funcției, care poate fi „TABLE", „VIEW", „RELATION" sau „CONNECTION". În exemplul nostru s-a optat pentru tabele.

Producătorul mediului Visual FoxPro recomandă înlocuirea (acolo unde este posibil) a macrosubstituției (considerată lentă) cu „numele de expresii" (*named expressions*). Acestea reprezintă o convenție de includere a unei variabile de tip șir de caractere între paranteze, la execuție evaluându-se nu șirul de caractere, ci variabila sau constanta reprezentată de acesta, ca în listingul 4.10 bis. Diferența este linia scrisă cu caractere **bold**.

Listing 4.10 bis. Deschiderea tuturor tabelelor bazei de date curente (varianta 2)

```
* exemplu de cod generalizat pentru
* deschiderea tabelelor unei baze de date
* varianta cu „named expression"
CLOSE TABLES ALL
clear
* extragem din containerul bazei de date(este tot o tabela) numele tabelelor
OPEN DATABASE "k:\aplicatie\database\vinzari" SHARED
*functia ADBOBJECTS populeaza un vector cu obiectele bazei de date curente
ADBOBJECTS(vTabele, "TABLE")
FOR i= 1 TO ALEN(vTabele,1)
    USE (vTabele(i)) IN 0
ENDFOR
```

Alt exemplu:

```
strAstazi="Astazi este " + dtoc(date())
?(strAstazi)
```

În fine, oarecum asemănătoare macrosubstituției este și funcția `EVALUATE(<expresie-de-tip-caracter>)`. De data aceasta expresia trebuie scrisă între ghilimele:

```
strAstazi="Astazi este " + dtoc(date())
?evaluate("strAstazi")
```

Alte exemple în care macrosubstituția constituie o soluție salvatoare vor fi prezentate în capitolul 5, consacrat dialectului SQL din Visual FoxPro, precum și în capitolul 7, partea rapoartelor dinamice.

4.7. Importul și exportul de date

Instrucțiunile pentru acces la fișiere de tip bază de date au fost tratate în capitolele ulterioare. Visual FoxPro are și facilități pentru lucrul cu fișiere de tip text (ASCII) și fișiere binare.

Accesarea fișierelor ASCII în Visual FoxPro

Pentru afișarea unor fișiere text se poate utiliza comanda TYPE <nume-fisier.extensie>, care este asemănătoare ca efect cu comanda TYPE din MS-DOS. Comenzile

```
COPY TO <nume-fișier> TYPE <tip-fișier>
```

și

```
APPEND FROM <nume-fișier[.extensie]> [TYPE <tip-fișier>]
```

se pot folosi pentru exportul/importul de date din tabele .dbf către o multitudine de formate, printre care și cel text (ASCII).

Dispunem de un fișier ASCII prezentat în figura 4.17.

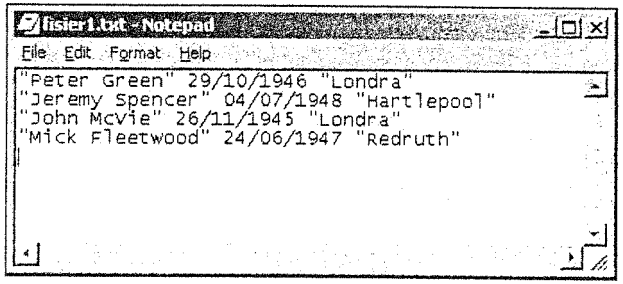


Figura 4.17. Fișier ASCII de tip *delimited with blank*

Acest fișier are o structură „tradițională" formată din câmpuri, dintre care cele de tip șir de caractere sunt încadrate de ghilimele. Este un format foarte răspândit, alături de cel CSV (*comma-separated values*) – vezi figura 4.18, la care rolul de delimitator este jucat de virgulă, iar prima linie conține numele câmpurilor.

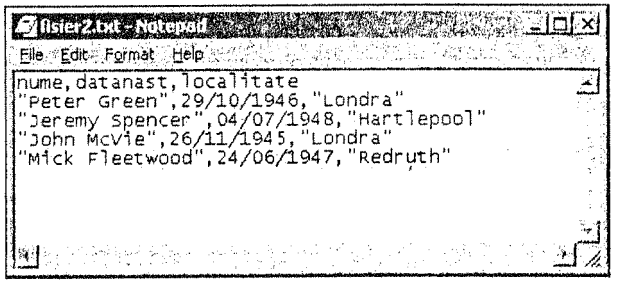


Figura 4.18. Fișier ASCII de tip CSV (*comma-separated values*)

Avem nevoie să importăm aceste date într-o tabelă, a cărei structură am creat-o în prealabil și care arată astfel:

```
NUME, C, 40
DATANAST, D (8 octeți, dar nu se scrie)
LOCALITATE, C, 20
```

Vom concepe, în vederea importului, programul din listingul 4.11.

Listing 4.11

```
* program pentru importul din fisiere ASCII
*******************************************************
* deschidem baza de date
IF !DBUSED("vinzari")
    OPEN DATABASE "k:\database\vinzari" SHARED
ENDIF
* deschidem tabela de test
IF !USED("teste_diverse")
    USE teste_diverse IN 0 SHARED
ENDIF

SELECT teste_diverse
* importam textul din fisierul ASCII
APPEND FROM "k:\fisier1.txt" DELIMITED WITH BLANK
* adaugam inca o persoana
APPEND BLANK
REPLACE nume WITH "Christine Perfect",;
    datanast WITH {^1943-07-12}, ;
    localitate WITH "Grenodd"
* afisam rezultatul
BROWSE TITLE "Apasati ESC pentru continuare..."
* inchidem tabela
USE
*******************************************************
```

Dacă se dorește importul din cea de-a doua variantă de fișier (numele fiind fisier2.txt), linia marcată cu caractere **bold** se va înlocui cu linia:

```
APPEND FROM "k:\fisier2.txt" TYPE CSV
```

Oricare ar fi varianta de program, rezultatul arată astfel:

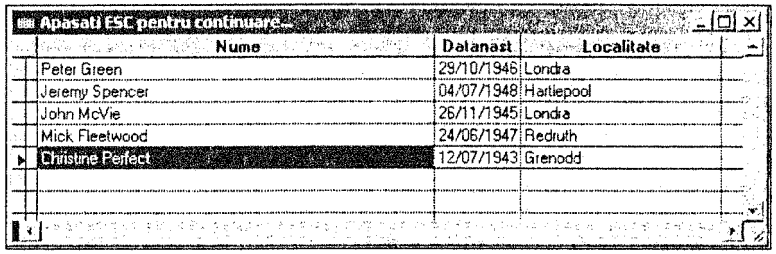


Figura 4.19. Rezultatul comenzii `APPEND FROM` la importul unui fişier text

Avem nevoie să exportăm aceste date într-un alt format, să zicem o foaie de calcul Excel 5. Această problemă se rezolvă prin comanda `COPY TO`, după cum se arată în programul din listingul 4.12:

Listing 4.12

```
* program pentru exportul datelor dintr-o tabela
*******************************************************
* deschidem baza de date
IF !DBUSED("vinzari")
    OPEN DATABASE "k:\database\vinzari" SHARED
ENDIF
* deschidem tabela de test
IF !USED("teste_diverse")
    USE teste_diverse IN 0 SHARED
ENDIF

SELECT teste_diverse
* exportam primele 3 inregistrari
GO TOP && prima inregistrare
COPY TO „k:\export1.xls" type XL5 next 3
USE
*******************************************************
```

Rezultatul arată ca în figura 4.20 (sunt exportate numele câmpurilor pe prima linie).

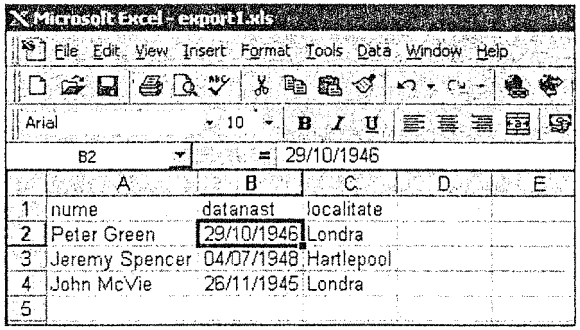


Figura 4.20. Rezultatul exportului către o foaie de calcul Excel

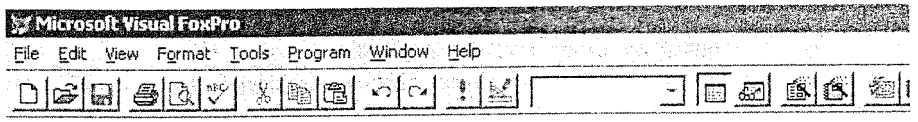
Pentru crearea de fişiere text există şi varianta folosirii comenzilor „\", respectiv „\\". Deosebirea dintre cele două variante este aceeaşi ca la comenzile „?" şi „??". Comenzile \ şi \\ se utilizează în combinaţie cu comanda `SET TEXTMERGE`. Astfel, dacă vom lansa

comanda `SET TEXTMERGE ON`, variabilele incluse între semnele << >> vor fi evaluate şi înlocuite cu valoarea lor; în caz contrar, vor apărea în text ca atare. Comanda `SET TEXTMERGE TO <nume-fişier.txt>` are ca efect redirecţionarea ieşirii comenzii (trimiterea textelor în fişierul text specificat în loc de ecranul principal Visual FoxPro). Anularea acestui comportament se face cu `SET TEXTMERGE TO` (care închide fişierul) şi `SET TEXTMERGE OFF` (care începe evaluarea variabilelor dintre semnele << >>).

Listing 4.13

```
* program de test pentru interclasare
CLEAR
if !DBUSED('vinzari') && daca baza de date nu-i deschisa
    OPEN DATABASE 'database\vinzari' SHARED && cale relativa: BD vinzari
                                              && din directorul database
ENDIF
IF !USED('clienti')          && daca tabela clienti nu-i deschisa
    USE clienti IN 0 SHARED
ENDIF
SELECT clienti

SET TEXTMERGE ON
SCAN FOR RECNO()<=4 && parcurgem primele 4 linii ale tabelei
\Clientul ce are codul
\\ <<ALLTRIM(str(codcl))>>
\\ se numeste <<ALLTRIM(dencl)>>
\\ cu sediul pe
\\ <<ALLTRIM(adresa)>>, cod <<ALLTRIM(codpost)>>

ENDSCAN
SET TEXTMERGE OFF
CLOSE ALL &&inchide tabelele si baza de date
```

Programul foloseşte şi funcţiile `DBUSED(<nume-bază-de-date>)` şi `USED(<nume-tabelă>)`, care permit să verificăm dacă o bază de date, respectiv o tabelă, sunt sau nu deschise. Şi încă ceva: operatorul logic `NOT` în Visual FoxPro se poate înlocui cu semnul exclamării „!", deci `!USED()` este acelaşi lucru cu `NOT USED()`. Iată rezultatul, în figura 4.21:



Clientul ce are codul 1001 se numeste Client 1 SRL cu sediul pe Tranzitiei, 13 bis, cod 6600
Clientul ce are codul 1002 se numeste Client 2 SA cu sediul pe , cod 6600
Clientul ce are codul 1003 se numeste Client 3 SRL cu sediul pe Prosperitatii, 22, cod 6500
Clientul ce are codul 1004 se numeste Client 4 cu sediul pe Sapientei, 56, cod 5725

Figura 4.21. Rezultatele execuţiei programului din listingul 4.13

Dacă linia scrisă în **bold** (vezi listingul 4.13) se înlocuieşte cu liniile:

```
SET TEXTMERGE ON
SET TEXTMERGE TO 'iesire.txt'
```

în loc de un rezultat afişat pe ecran se va obţine fişierul `iesire.txt`, în directorul curent, cu exact acelaşi conţinut.

Comenzile SET TEXTMERGE, „\" şi „\\" merită a fi luate în seamă atunci când dorim să efectuăm o interclasare (ca de pildă trimiterea unei scrisori cu conţinut standard către toţi clienţii). Comenzile COPY TO..., APPEND FROM..., \ şi \\ sunt utile atunci când avem nevoie să facem schimb de date cu aplicaţii de pe alte platforme. Formatul ASCII este universal, pe când formatele .dbf, .xls, .wks şi altele similare nu sunt.

4.7.1. Accesarea fişierelor binare în Visual FoxPro. Recuperarea datelor din tabele cu antetul (*header*-ul) eronat

Veţi avea probabil nevoie de asemenea facilităţi în cazul în care doriţi să realizaţi o aplicaţie mai sofisticată. Ca şi un fişier ASCII, un fişier binar este un şir de octeţi pe disc, dar spre deosebire de primul, în care fiecare octet reprezintă un caracter (literă sau cifră), aici octeţii au semnificaţii diferite. În Visual FoxPro, un număr se memorează pe 8 octeţi (similar cu tipul Windows DWORD – double din C++ sau Basic), o dată de tip logic pe 1 octet, un şir de caractere pe atâţia octeţi câte caractere are etc.

În citirea/scrierea unui fişier binar, de o importanţă capitală sunt pointerul şi deplasamentul (*offset*). Pointerul este un fel de indicator ce arată poziţia în fişier, iar offset-ul, un număr de octeţi număraţi de la începutul fişierului. Astfel, dacă ştim că un fişier binar începe cu un număr şi dorim să-l citim, trebuie să avansăm cu 8 octeţi printr-o instrucţiune adecvată (operaţiile de citire-scriere produc şi avansul pointerului). În Visual FoxPro, fişierele binare sunt accesate printr-un set de funcţii numite funcţii pentru acces *low-level*.

Funcţia FOPEN(<nume-fişier>[,atribut]) returnează un număr ce poate fi folosit ulterior deschiderii fişierului specificat, ca şi *alias* pentru fişier şi care în paginile următoare va fi referit ca număr-fişier (denumirea consacrată este *file handler*). Dacă fişierul nu a putut fi deschis, atunci rezultatul acestei funcţii este –1.

Tabelul 4.2. Moduri de deschidere a unui fişier binar

Atribut	*Drepturi*	*Mod de citire/scriere a datelor*
0	Read Only (implicit)	cu buffer
1	Write-Only	cu buffer
2	Read / Write	cu buffer
10	Read-Only	fără buffer
11	Write-Only	fără buffer
12	Read / Write	fără buffer

Dacă nu se specifică o valoare pentru atribut, se consideră 0, deci „mod de acces read-only cu buffer".

Funcţia FCREATE((<nume-fişier>[,atribut]) are rolul de a crea (şi deschide) un fişier cu numele specificat. Dacă acelaşi fişier există deja, el este distrus. Argumentul atribut poate lua una dintre valorile din tabelul de mai jos, cu semnificaţiile respective.

Tabelul 4.3. Atribute posibile la crearea unui fişier binar

Atribut	*Atributele fişierului creat*
0	Read-Write (implicit)
1	Read-Only
2	Hidden
3	Read-Only/Hidden
4	System

Atribut	*Atributele fişierului creat*
5	Read-Only/System
6	System/Hidden
7	Read-Only/Hidden/System

O dată fişierul deschis, putem scrie în el cu ajutorul funcţiilor:

FWRITE(număr-fişier,expresie[,număr- octeţi-de-scris])

sau

FPUTS(număr-fişier,expresie[,număr- octeţi-de-scris]).

Ambele funcţii returnează un număr ce înseamnă numărul de octeţi scrişi în fişier (0 dacă scrierea nu a putut avea loc). FPUTS scrie ca ultim caracter o combinaţie carriage-return+formfeed (CRLF sau avans de linie, care în Visual FoxPro se simulează prin CHR(13)+CHR(10)).

Pentru citire se pot folosi funcţiile:

FREAD(număr-fişier,număr-octeţi-de-citit)

sau

FGETS(număr-fişier[,număr-octeţi-de-citit]).

A doua funcţie citeşte octeţii până la perechea de caractere CRLF, dacă nu s-a specificat altfel prin argumentul număr-octeţi-de-citit.

Deplasarea într-un fişier binar are loc cu ajutorul funcţiei:

FSEEK(număr-fişier, deplasament-ca-nr-de-octeţi [, poziţie-relativă])

care „sare" un număr specificat de octeţi, începând de la poziţia indicată prin cel de-al treilea argument. Acesta poate avea una dintre valorile 0 (începutul fişierului), 1 (poziţia curentă) sau 2 (sfârşitul fişierului). Rezultatul funcţiei arată deplasamentul (în octeţi) faţă de începutul fişierului.

Funcţia FEOF(număr-fişier) returnează .T. sau .F., după cum ne aflăm sau nu la sfârşitul fişierului. Utilă este şi funcţia FFLUSH(număr-fişier), ce are misiunea de a scrie informaţiile din buffer pe disc, în cazul accesului „bufferizat" la fişierul binar. În fine, funcţia FCLOSE(număr-fişier) realizează închiderea fişierului specificat.

Utilitatea acestor comenzi pentru un programator de FoxPro se poate dovedi în mai multe cazuri. Vom analiza aici o problemă legată de coruperea formatului unui fişier .DBF, care face imposibilă deschiderea sa cu USE.

Dispunem de tabela teste_diverse.DBF, a cărei structură ştim că arată astfel:

```
NUME, C, 40
DATANAST, D (8 octeţi)
LOCALITATE, C, 20
```

Header-ul unei tabele .DBF (zonă cu metadate) memorează multe informaţii importante (numele bazei de date de care aparţine tabela, numărul de câmpuri, numărul de înregistrări, numele şi ordinea câmpurilor etc.). Presupunând că *header*-ul unei tabele a fost compromis accidental şi, ca urmare, comanda USE nu mai funcţionează, ne revine misiunea de a recupera cât mai mult din datele rămase în tabelă. Acest lucru nu este imposibil cu puţin exerciţiu în folosirea funcţiilor *low-level*. Trebuie să cunoaştem structura tabelei (câmpurile, cu tipurile şi lungimile lor) şi câte ceva despre formatul unui fişier .DBF (de mare ajutor aici ne este sistemul de *help* al Visual FoxPro; Programmer'sGuide→Appendix→TableFileStructure). Aflăm de aici că *header*-ul unei tabele are o lungime de *296 octeţi+32 octeţi*numărul de câmpuri din structură.*

În cazul nostru sunt 3 câmpuri, deci *header*-ul tabelei are o lungime de 296+32*3=392 octeți. Datele propriu-zise vor începe de la octetul 393. Firește, în loc de toate aceste calcule am fi putut utiliza funcția HEADER (<nume-tabelă>), care returnează chiar dimensiunea în octeți a *header*-ului tabelei, dar reamintiți-vă că aceasta funcționează doar pe tabele valide, care se pot deschide cu USE.

Secvența de cod pe care o vom prezenta în listingul 4.14 deschide fișierul în mod binar, sare peste primii 393 de octeți (mărimea calculată a *header*-ului), execută citirea câmp cu câmp și afișează datele citite direct pe ecranul principal Visual FoxPro.

Listing 4.14. Program pentru citirea unor date în format binar

```
* incercare de recuperare a datelor dintr-o
* tabela .DBF compromisa
* numele si structura tabelei:
* TESTE_DIVERSE(nume, C(40), datanast D, localitate C(20))
CLEAR
* deschidem fisierul (extensia e obligatorie)
nTabela=FOPEN("k:\database\teste_diverse.dbf")
IF nTabela=-1
    MESSAGEBOX("Nu pot deschide fisierul. Exista sau e cumva deschis deja?")
ELSE
* deplasare 392 de octeti (296+3cimpuri*32octeti per camp)
FSEEK(nTabela,392)
* in fiecare inregistrare primul octet
* constituie marcajul de stergere
marcaj=FREAD(nTabela,1)
* citim tabela pana la sfarsit
DO WHILE NOT FEOF(nTabela)
        camp1=FREAD(nTabela,40)
        camp2=FREAD(nTabela,8)
        camp3=FREAD (nTabela, 20)
        ? marcaj, camp1
        ?? DATE(VAL(LEFT(camp2,4)),VAL(SUBSTR(camp2, 5,2)), VAL(SUBSTR(camp2,7,2)))
        ?? " ", camp3
* citim primul octet de pe inregistrare: dupa ultima inregistrare
* el nu mai este marcaj de stergere, ci caracterul EOF
        marcaj=FREAD(nTabela,1)
ENDDO
* inchidem fisierul
FCLOSE(nTabela)
ENDIF
```

Rezultatul execuției este prezentat în figura 4.22. Asteriscul de lângă prima înregistrare arată marcare pentru ștergere. De observat că în cazul câmpului de tip dată calendaristică s-a efectuat o retratare cu ajutorul funcțiilor pentru lucrul cu șiruri de caractere: LEFT() și SUBSTR(), funcției de conversie din șir de caractere în număr VAL() și funcției DATE(an, lună, zi). Acest lucru a fost necesar deoarece în fișier, data calendaristică se stochează ca un șir de octeți în forma AAAALLZZ.

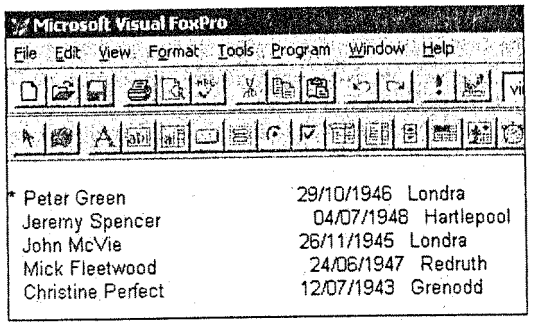


Figura 4.22. Date recuperate din fișierul teste_diverse.dbf

Deocamdată afișarea a avut loc pe ecran. Cu titlu de exercițiu, încercați să „treceți pe curat" (într-o tabelă cu structură similară) datele recuperate.

4.8. Tratarea erorilor în programele Visual FoxPro

Putem spune, pe scurt, că erorile dintr-un program sunt de două mari categorii: *erori de sintaxă* (pot fi depistate deoarece împiedică etapele de compilare și de execuție a programului) și *erori de execuție*, care nu pot fi descoperite înainte de execuția programului (pentru detecția lor, programul se execută folosind *date de test*, adică date pentru care rezultatele prelucrărilor sunt dinainte cunoscute).

Testarea este activitatea de descoperire a erorilor, iar *depanarea* este operația de izolare și corectare a acestora.

În realizarea unei aplicații, programatorul poate avea în vedere diferite niveluri ale robustеții acesteia:

- execuția fără blocaje și mesaje de eroare derutante pentru utilizatorul final;
- acțiuni adecvate în scenariile cele mai frecvente;
- posibilitatea unei acțiuni rezonabile sau oferirea unui mesaj de eroare edificator pentru o varietate de scenarii;
- capacitatea restabilirii aplicației după o acțiune neașteptată a utilizatorului.

Prudența este mama înțelepciunii, așa că, înainte de a aștepta ca erorile să apară la locul de exploatare a aplicației, realizatorul acesteia trebuie să o verifice, să elimine erorile evidente și să reducă pe cât posibil riscul apariției de noi erori. Studiile au arătat că o practică disciplinată de scriere a codului-sursă (folosirea comentariilor și a liniilor libere, respectarea unor convenții pentru denumirea variabilelor și constantelor etc.) duce la reducerea semnificativă a numărului de erori.

În al doilea rând, o cale de a preveni erorile la execuție este codificarea algoritmilor de o așa manieră încât înaintea unei comenzi să aibă loc o verificare minimală a contextului acesteia. De exemplu, comada SKIP generează o eroare dacă se întâlnește sfârșitul tabelei. Dar, pentru ca o astfel de eroare să nu apară niciodată, programatorul poate folosi o secvență de program de forma următoare:

```
IF !EOF()
    SKIP
```

```
ELSE
    GO BOTTOM
ENDIF
```

Visual FoxPro, ca orice mediu de dezvoltare din categoria sa, dispune de mecanisme implicite de semnalare a erorilor şi de corecţie a acelor erori care nu sunt fatale. Totuşi, multe dintre erorile Visual FoxPro cer, atunci când survin într-un program, oprirea imediată a acestuia. Dar dacă ar exista o procedură care să intre în acţiune automat şi care, în funcţie de natura erorii, să ofere, prin structuri de control adecvate, căi (mai flexibile) de rezolvare a acesteia ? Este imposibil a cunoaşte dinainte multitudinea acestor situaţii. Programatorul însă poate acţiona în direcţia „adaptării la problemă", a manipulării unor erori pe care le anticipează (lipsa unor fişiere de date, absenţa unor drepturi de acces etc.). Deci Visual FoxPro este dotat cu mecanisme implicite de captare a erorilor, cu un mare caracter de generalitate, iar programatorului de aplicaţii îi revine sarcina de a concepe şi pune în practică propriile sale module de tratare a erorilor.

Acest demers al programatorului este favorizat de existenţa instrucţiunii ON ERROR, a cărei sintaxă completă este ON ERROR [comandă] şi se plasează de regulă la începutul programelor pentru care dorim o tratare „personalizată" a erorilor. În cazul apariţiei unei erori pe parcursul execuţiei programului, controlul se transferă instrucţiunii de după ON ERROR şi nu mai intră în acţiune mecanismul implicit de semnalare a erorilor. Efectul acestei instrucţiuni se propagă de la modulul apelant către modulul apelat, deci în cazul conceperii unei rutine de tratare a erorilor pentru o aplicaţie cu mai multe module de program, această instrucţiune se plasează în programul principal.

Notă. ON ERROR are efect şi în fereastra de comenzi.

Presupunând că procedura destinată tratării erorilor se numeşte trat_erori, comanda pentru activarea sa ar trebui să arate astfel:

```
ON_ ERROR DO trat_erori
```

Dacă vom scrie doar ON ERROR, fără nimic altceva, atunci va avea loc reactivarea mecanismului implicit de manipulare a erorilor din Visual FoxPro.

După cum se arată în figura 4.23, se poate provoca intenţionat o eroare prin scrierea unei comenzi greşite.

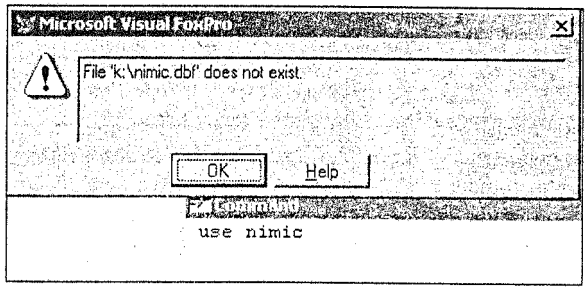


Figura 4.23. Provocarea intenţionată a unei erori

De asemenea, se poate folosi instrucţiunea:

ERROR număr-eroare [,parametru eroare]|[mesaj eroare]

pentru a genera o eroare Visual FoxPro al cărei număr se cunoaşte (documentaţia Visual FoxPro specifică şi care sunt codurile de eroare). Putem folosi această instrucţiune în scopuri de test, ca de exemplu:

ERROR 13 – va afişa fereastra din figura 4.24 stânga;

ERROR 13, „participatii" – va afişa mesajul din dreapta aceleiaşi figuri.

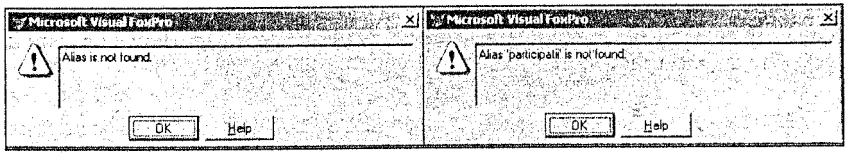


Figura 4.24. Erori generate de utilizator

Listing 4.14 bis. Procedura de tratare a erorilor

```
* program demonstrativ pentru
* simularea erorilor
ON ERROR DO "k:\aplicatie\progs\trat_erori.prg"
USE nimic IN 0 SHARED && stim ca nu exista tabela "nimic"
ON ERROR
```

Urmează procedura propriu-zisă de tratare a erorilor, redusă la cea mai simplă expresie:

```
MESSAGEBOX("A aparut o eroare!")
```

La lansarea programului simulare_erori prin comanda DO k:\aplicatie\simulare_erori.prg va apărea mesajul din figura 4.25.

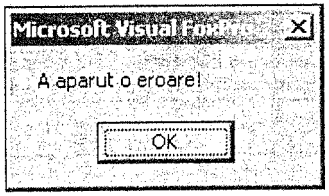


Figura 4.25. Mesajul afişat de procedura trat_erori

Exemplul prezentat este pe cât de simplu, pe atât de inutil. Că a apărut o eroare am fi aflat şi fără să fi folosit această instrucţiune, şi tot n-am fi putut lua măsuri. În cazul de faţă ar fi fost indicat să cerem de la utilizator precizarea tabelei de deschis şi să continuăm lucrul. Dar nu ştim ce anume a cauzat eroarea: să fie comanda DO, să fie comanda USE ? E util să ştim că, în caz de utilizare a instrucţiunii ON ERROR [comandă], putem afla informaţii suplimentare de la următoarele funcţii:

- ERROR() – furnizează numărul erorii. Pentru o listă exhaustivă a numerelor de eroare, se va consulta documentaţia ataşată produsului Visual FoxPro (chiar şi *help*-ul în format electronic, la rubrica Reference→Error Messages);

- `MESSAGE()` – returnează textul mesajului de eroare (descrierea erorii). Pentru detalii privind multitudinea mesajelor de eroare, sunt valabile observațiile privind sistemul de *help*. O cale rapidă de aflare a descrierii unei erori este tastarea unei comenzi voit eronate în fereastra de comenzi;
- `LINENO([1])` – obține numărul liniei curente de program, începând cu prima linie a programului curent (sau, dacă se scrie `LINENO(1)`, începând cu prima linie a programului principal).

Notă. În numărătoarea liniilor sunt incluse și liniile de comentarii, liniile de continuare (cele care urmează caracterului „punct și virgulă") și liniile vide.

- `PROGRAM([nivel-program])` sau `SYS(16 [, nivel-program])` returnează numele programului în curs de execuție. Dacă nu se specifică argumentul nivel-program, se obține numele programului curent. Dacă se specifică 0 sau 1, se obține numele programului principal, iar dacă se specifică altă cifră (până la 128), se returnează numele programului de la nivelul respectiv (vezi figura 4.26).

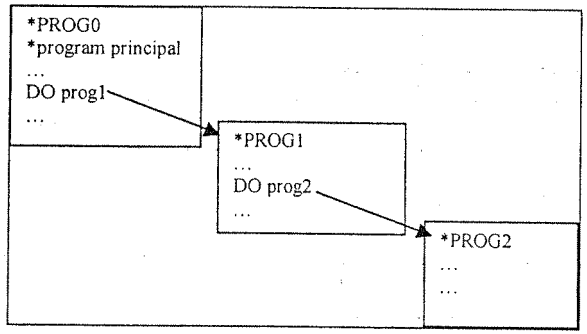


Figura 4.26. Niveluri de imbricare a programelor Visual FoxPro

Vom rescrie cele două programe ca în listingurile 4.15 și 4.15 bis.

Listing 4.15. Programul de test

```
* simulare_erori
* program demonstrativ pentru
* simularea erorilor
ON ERROR DO "k:\aplicatie\progs\trat_erori.prg" ;
    WITH ERROR(), MESSAGE(), PROGRAM(), LINENO()
USE nimic IN 0 SHARED  && stim ca nu exista tabela "nimic"
ON ERROR                        && dupa terminare, dezactiveaza tratarea erorilor
```

Listing 4.15 bis. Procedura de tratare a erorilor

```
* trat_erori
* procedura de tratare a erorilor
* cateva definitii de constante pentru functia MESSAGEBOX()
PARAMETERS nreroare, descriere, numeprogram, nrlinie
#DEFINE CRLF                    CHR(10)+CHR(13)  && ENTER
#DEFINE MBOK                    0           && numai butonul OK
```



```
#DEFINE MBOKCANCEL          1       && OK si Cancel
#DEFINE MBEXCLAMATION       48      && semnul exclamarii
#DEFINE TITLU "Eroare captata de utilizator"
* semnalarea erorii
MESSAGEBOX("Eroare nr. " +ALLT(STR(nreroare)) + ". " + ;
    descriere +CHR(13)+CHR(10)+ ;
    " in modulul " + numeprogram + ", pe linia " + ;
    ALLT(STR(nrlinie)) + "." , ;
    MBOK+MBEXCLAMATION, TITLU)
```

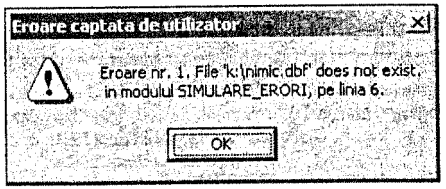


Figura 4.27. Indicarea detaliată a erorii

Mesajul afișat de subrutina de tratare a erorilor este de data aceasta ceva mai concludent (figura 4.27).

Oarecum similară folosirii celor patru funcții de mai sus este utilizarea funcției `AERROR(<nume-tablou>)`, care populează cu informații detaliate privind eroarea un tablou cu 7 coloane. În cazul apariției unei erori „native" Visual FoxPro, tabloul are o singură linie (adică este vector). Semnificațiile elementelor sunt prezentate în tabelul 4.4.

Tabelul 4.4. Elementele vectorului de erori

Număr coloană	*Descriere*
1	Numeric. Codul erorii (similar funcției `ERROR( )`).
2	Caracter. Textul mesajului de eroare (similar funcției `MESSAGE( )`).
3	`NULL`. Totuși, dacă eroarea are un parametru suplimentar[24], conține textul acestui parametru (similar funcției `SYS(2018)`).
4	`NULL`. Dacă eroarea se referă la o tabelă deschisă, conține numărul respectivei zone de lucru .
5	`NULL`. Totuși, dacă eroarea este `1539` – „trigger failed", conține una dintre următoarele valori numerice: 1 – a eșuat declanșatorul `Insert`; 2 – a eșuat declanșatorul `Update`; 3 – a eșuat declanșatorul `Delete`.
6	`NULL`.
7	`NULL`.

Folosind un vector pentru preluarea erorilor, putem modifica fragmentele de cod prezentate mai sus ca în listingurile 4.16 și 4.16 bis.

24. De exemplu, un nume de fișier, un nume de variabilă etc.

Listing 4.16. Programul de test

```
* simulare_erori versiunea 2
* program demonstrativ pentru
* simularea erorilor
ON ERROR DO "k:\aplicatie\progs\trat_erori.prg" ;
    WITH PROGRAM(), LINENO()
USE nimic IN 0 SHARED          && stim ca nu exista tabela "nimic"
ON ERROR                       && dupa terminare, dezactiveaza tratarea erorilor
```

Listing 4.16 bis. Procedura de tratare a erorilor

```
* trat_erori versiunea 2
* procedura de tratare a erorilor
PARAMETERS numeprogram, nrlinie
#DEFINE CRLF                CHR(10)+CHR(13)  && ENTER
#DEFINE MBOK                0          && numai butonul OK
#DEFINE MBOKCANCEL          1          && OK si Cancel
#DEFINE MBEXCLAMATION       48         && semnul exclamarii
#DEFINE TITLU "Eroare captata de utilizator"
* preluarea erorii
AERROR(vErori)                         && se creeaza automat vectorul „vErori"
* semnalarea erorii
MESSAGEBOX("Eroare nr. " +ALLT(STR(vErori(1,1))) + ". " + ;
    vErori(1,2) +CHR(13)+CHR(10)+ ;
    " in modulul " + numeprogram + ", pe linia " + ;
    ALLT(STR(nrlinie)) + "." , ;
    MBOK+MBEXCLAMATION, TITLU)
```

Am recurs totuşi la o variantă combinată `AERRORS()`, `PROGRAM()` şi `LINENO()`, deoarece vectorul de erori nu precizează nimic în legătură cu programul şi linia la care a apărul eroarea. Este adevărat că aceste informaţii îl privesc mai mult pe programator, în etapa de depanare a programului, şi vor avea foarte puţină relevanţă pentru un utilizator final; de aceea nici nu se recomandă oferirea lor într-o versiune finală a aplicaţiei.

Faptul că informăm utilizatorul despre o eroare nu-l ajută cât ar dori acesta. Trebuie să-i oferim şi o cale de acţiune. Pentru aceasta, considerând tot cazul de mai sus, va trebui să disociem eroarea nr. 1 (lipsă fişier) de alte erori şi să-i aplicăm un tratament special. Vom reda în continuare numai varianta modificată a procedurii `trat_erori`.

Listing 4.17. Varianta îmbunătăţită a tratării erorilor

```
* trat_erori versiunea 3
* procedura de tratare a erorilor
*************************************************************************
PARAMETERS numeprogram, nrlinie
#DEFINE CRLF                CHR(10)+CHR(13)  && ENTER
#DEFINE MBOK                0          && numai butonul OK
#DEFINE MBOKCANCEL          1          && OK si Cancel
#DEFINE MBRETRYCANCEL       5          && Retry si Cancel
#DEFINE MBQUESTION          32         && semnul intrebarii
#DEFINE MBEXCLAMATION       48         && semnul exclamarii
#DEFINE IDOK                1          && raspuns OK
#DEFINE IDRETRY             4          && raspuns Retry
#DEFINE IDCANCEL            2          && raspuns Cancel
#DEFINE TITLU "Eroare captata de utilizator"
* preluarea erorii
AERROR(vErori)                         && se creeaza automat vectorul vErori
```



```
* semnalarea erorii
DO CASE
CASE vErori(1,1)=1                                  && file does not exist
    IF IDRETRY=MESSAGEBOX("Nu gasesc fisierul " +vErori(1,3)+"." +CRLF + ;
              "Il cautati (Retry) sau abandonati (Cancel)?", ;
              MBRETRYCANCEL +MBQUESTION,TITLU)
* functia GETFILE afiseaza dialogul Open (in care alegem un fisier)
        USE GETFILE("DBF", "Numele fisierului", "Deschide", 0, ;
              "Alegeti un fisier .DBF:") IN 0 SHARED
    ENDIF
OTHERWISE                                           && eroare netratata (inca)
    MESSAGEBOX("Eroare nr. " +ALLT(STR(vErori(1,1))) + ". " + ;
        vErori(1,2) +CHR(13)+CHR(10)+ ;
        " in modulul " + numeprogram + ", pe linia " + ;
        ALLT(STR(nrlinie)) + "." , ;
        MBOK+MBEXCLAMATION, TITLU)
ENDCASE
*************************************************************************
```

Într-o procedură de acest tip, programatorul poate completa ramurile structurii `DO CASE...ENDCASE` cu alte secvenţe de cod care să permită tratarea erorii în funcţie de codul acesteia.

4.9. Depanarea programelor cu instrumentul Debugger

Visual FoxPro dispune de un instrument sofisticat pentru depanarea programelor, numit *Debugger* (depanator). Acesta se apelează fie din meniul `Tools→Debugger`, fie în mod direct, prin lansarea comenzii `DEBUG`. Indiferent ce mod de apelare veţi folosi, va trebui să lucraţi într-un sistem de ferestre cu aspectul din figura 4.28.

Vom explica pe rând rolul fiecăreia dintre ferestrele vizibile în figură (care pot fi ascunse/afişate şi cu butoanele marcate cu cifre între (8) şi (12)):

- `Trace` – afişează codul-sursă în curs de testare;
- `Watch` – prezintă valoarea la un moment dat a unor expresii a căror „urmărire" a fost specificată anterior de către programator (prin scriere în căsuţa de text `Watch` sau prin drag&drop din orice fereastră care ar conţine expresia dorită);
- `Call Stack` – prezintă numele programelor executate, în ordinea apelării (modulul cel mai recent apelat este primul în listă); dacă nu sunt module apelate, atunci în această fereastră vom avea numele unui singur program;
- `Debug Output` – afişează mesajele generate de utilizarea în programe a instrucţiunii `DEBUGOUT <expresie>`; această instrucţiune se poate utiliza pentru a marca trecerea printr-o anumită secţiune a codului-sursă;
- `Locals` – permite supravegherea valorilor variabilelor locale din procedura în curs de testare.

Pentru a demonstra folosirea acestui puternic instrument, am recurs la cele două programe `simulare_erori`, respectiv `tratare_erori`. Am inclus în codurile-sursă ale programelor linia `DEBUGOUT „Lansare program „ + PROGRAM()` ca primă linie (ca a doua linie într-o procedură, deoarece prima trebuie să fie neapărat declaraţia `PARAMETERS`), pentru a urmări în fereastra `Debug Output` când se apelează

fiecare modul. Apoi am încărcat fişierul `simulare_erori.prg` în `Debugger`, prin click pe butonul marcat cu (1).

Lansarea în execuţie a programului încărcat are loc prin click pe butonul marcat cu (2), iar oprirea înainte de final, prin click pe butonul marcat cu (3) (vezi figura 4.28).

După cum se observă, în momentul execuţiei ferestrele depanatorului sunt foarte animate:

- în fereastra `Debug Output` s-au afişat cele două mesaje de la începutul fiecărui modul de program;

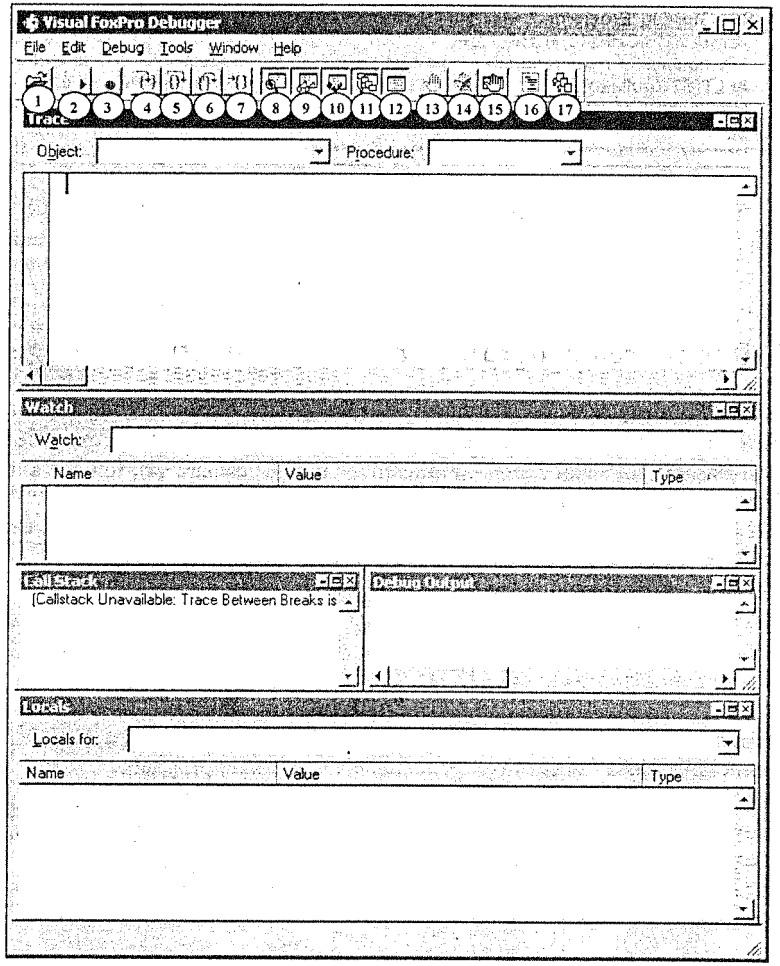


Figura 4.28. Ferestrele utilitarului `Debugger` (modul inactiv)

- în fereastra `Call Stack` o săgeată indică modulul în curs de execuţie;
- în fereastra `Locals` sunt afişate variabilele locale din modulul curent (vectorul `verori` a fost expandat pentru a consulta valorile elementelor sale);

- fereastra `Watch` nu conţine deocamdată nimic, deoarece nu s-a optat pentru urmărirea nici unei variabile;
- pe ecran s-a afişat o casetă de mesaj care provine din procedura de tratare a erorilor;
- în fereastra `Trace` se afişează codul-sursă al modulului curent, iar un vârf de săgeată indică linia în curs de execuţie.

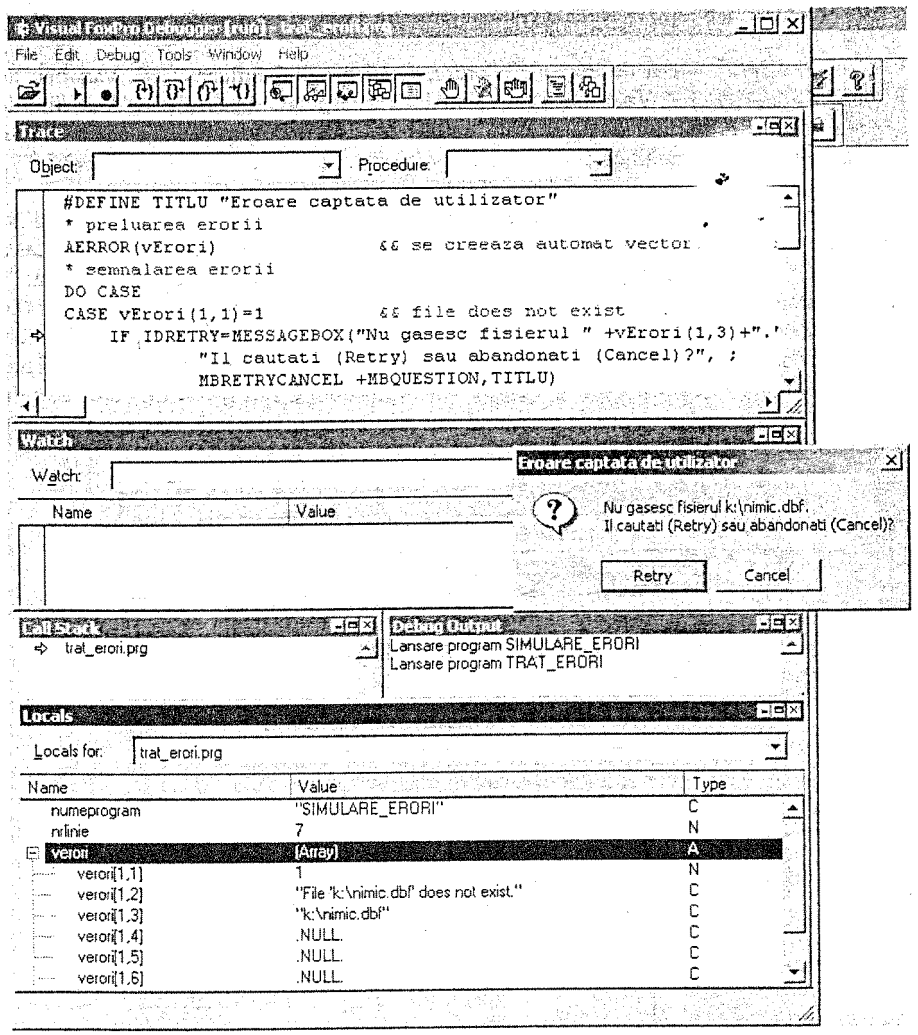


Figura 4.29. Program în curs de execuţie în utilitarul `Debugger`

În anumite cazuri se doreşte o execuţie pe porţiuni a programului (numai o procedură sau numai o funcţie). Pentru a face aceasta, există următoarele posibilităţi (vezi figura 4.28):

- `Step into` (4) – realizează execuţia pas cu pas a unei funcţii sau proceduri – la fiecare apăsare pe butonul (2) se execută câte o instrucţiune;
- `Step over` (5) – sare peste funcţia (procedura) curentă;
- `Step out` (6) – realizează ieşirea din funcţia (procedura) curentă;
- `Run to cursor` (7) – execută programul până la poziţia curentă a cursorului (care se stabileşte executând click în fereastra `Trace`, pe linia dorită).

Observaţie: viteza de execuţie a programului poate fi influenţată de variabila de mediu `_THROTTLE`. Valorii acestei variabile îi corespunde o pauză între două linii de cod succesive, variind de la 0 (fără pauză) la circa 5,5 secunde. Valoarea acestei variabile se poate modifica din fereastra de comenzi, de exemplu `_THROTTLE=0.5`.

Uneori e nevoie să suspendăm execuţia programului într-un anumit punct şi să verificăm anumite condiţii ori valoarea anumitor expresii. Într-o formă rudimentară, aceasta se poate face prin plasarea unei instrucţiuni `SUSPEND` în punctul dorit din codul-sursă. În atare condiţii, execuţia programului se opreşte şi dacă el este executat în mod normal (nu neapărat în depanator). După verificarea condiţiilor respective, programul poate fi reluat prin comanda `RESUME` (în fereastra de comenzi, dacă programul se execută în mod direct) ori prin butonul (2), dacă execuţia are loc în depanator. Oprirea execuţiei are loc prin comada `CANCEL` (în fereastra de comenzi) sau prin butonul (3), dacă execuţia are loc în depanator (vezi figura 4.28).

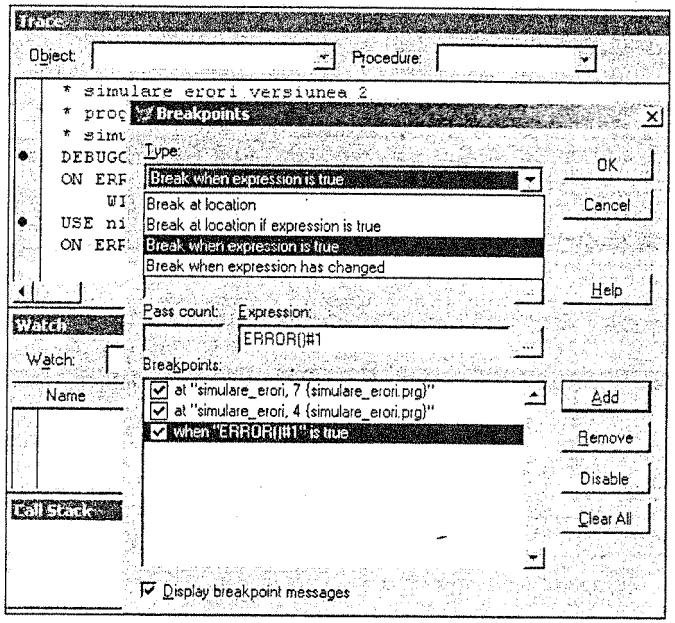


Figura 4.30. Dialogul `Breakpoints`

O modalitate de suspendare a execuţiei mult mai elegantă este stabilirea de puncte de oprire (*breakpoints*) în codul-sursă, prin poziţionarea cursorului în fereastra `Trace`, pe linia dorită şi prin click pe butonul `Toggle Breakpoint` (marcat cu [13] în figura 4.28). În marginea stângă a ferestrei `Trace` va apărea un punct roşu, în dreptul liniei respective. Acelaşi lucru se poate face şi prin dublu-click în chiar marginea stângă a ferestrei `Trace`, la linia dorită. O apăsare repetată a butonului (13) are ca efect anularea punctului de oprire, iar butonul marcat cu (14) permite anularea tuturor punctelor de oprire din program. Dacă se doreşte specificarea unor puncte de oprire sofisticate (care să intre în acţiune numai la întrunirea unor condiţii [`Break if...`, `Break when...`] ori la a *n*-a parcurgere, de exemplu), se poate folosi butonul `Breakpoint Dialog` (15), a cărui apăsare afişează o fereastră pe care o putem vedea în figura 4.30.

Evaluarea performanţei execuţiei aplicaţiilor Visual FoxPro se poate face cu ajutorul unei alte opţiuni accesibile din depanator, anume `Toggle Coverage Logging` (butonul 16 din figura 4.28). Acest buton este de tip „toggle" (comutator) – un click activează opţiunea, alt click o dezactivează. La activarea sa (numai dacă în `Debugger` este încărcat deja un program) apare pe ecran fereastra din figura 4.31, solicitându-se un nume de fişier.

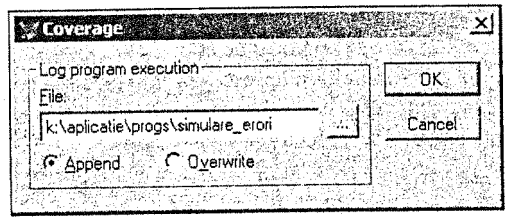


Figura 4.31. Stabilirea opţiunii `Coverage Logging`

Extensia fişierului principal este implicit `.log` (jurnal) şi, de fapt, acesta este şi rolul opţiunii: de a jurnaliza execuţia programului. Astfel, dacă în fereastra din figura 4.31 se execută click pe butonul `OK` iar apoi se lansează programul, se va obţine fişierul `simulare_erori.log`, care poate fi consultat ulterior, de pildă cu programul `Notepad`.

Atenţie! Pentru deschiderea fişierului `.log` cu `Notepad` este necesar ca acesta să fie închis de Visual FoxPro prin dezactivarea opţiunii `Coverage Logging` (verificaţi ca butonul să nu fie apăsat).

Activarea este echivalentă cu execuţia comenzii `SET COVERAGE TO <nume-fişier>`, dezactivarea – cu comanda `SET COVERAGE TO`.

Iată cum arată conținutul fișierului-jurnal:

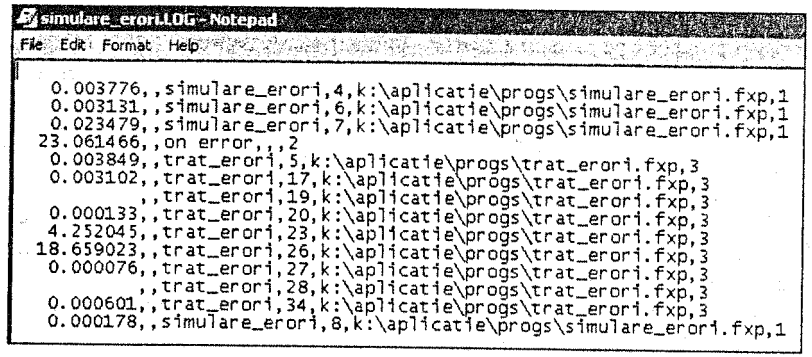


Figura 4.32. Jurnalul execuției unui program

Fișierul `.log` are atâtea rânduri câte linii de program s-au executat (incluzând modulele apelate). Elementele de pe o linie a acestui fișier au următoarele semnificații:

- durata de execuție, în secunde;
- numele clasei în care se află linia de cod (dacă e cazul);
- numele modulului în care se află linia de cod;
- numărul liniei de cod (liniile care lipsesc sunt linii albe sau comentarii);
- numele și calea completă a fișierului compilat din care face parte linia de cod;
- ordinea apelării modulului respectiv (`Call Stack`).

Execuția liniei 26 din procedura `trat_erori` a durat mai mult de 18 secunde! E normal ? Da, deoarece acea linie conține funcția `GETFILE()`, care afișează un dialog de tip `Open` și așteaptă până ce utilizatorul selectează un fișier. Și se pare că utilizatorul l-a selectat în 18,65 secunde...

O analiză mai elegantă decât simpla citire cu `Notepad` o realizează accesoriul `Coverage Profiler`, apelabil din meniul `Tools` al ferestrei principale Visual FoxPro. Încărcarea unui fișier `.log` se face prin `File→Open`. În bara de titlu (vezi figura 4.33) se observă fișierul `.log` creat mai înainte.

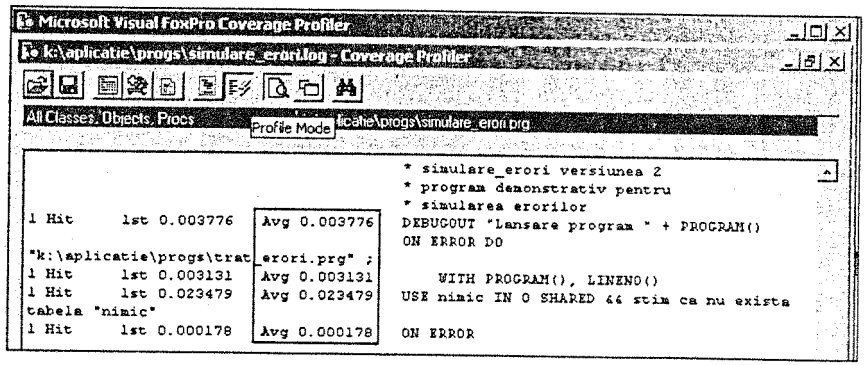


Figura 4.33. Instrumentul `Coverage Profiler` în modul `Profile` (1)

Se poate opta pentru vizualizarea simplă a fișierului (`Coverage Mode`) sau pentru afișarea unor indicatori de performanță (`Profile Mode`). Acești indicatori sunt mai concludenți dacă programul se execută de mai multe ori. Pentru aceasta, se închide fișierul `.log` (preferabil și instrumentul `Coverage Profiler`) și se revine la `Debugger`; în dialogul prezentat în figura 4.34 se alege același fișier `.log`, se stabilește opțiunea `Append` (adaugă la precedentul jurnal noile rezultate) și se repetă execuția programului. Apoi se relansează accesoriul `Coverage Profiler`.

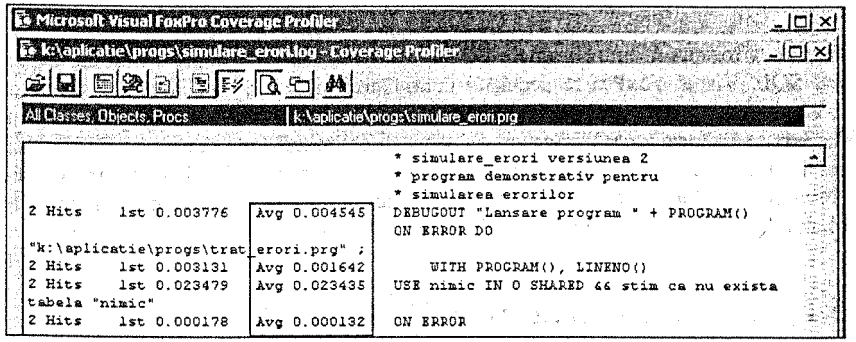


Figura 4.34. Instrumentul `Coverage Profiler` în modul `Profile` (2)

Se observă de această dată că:

- se arată numărul de execuții ale programului (2 Hits);
- valorile medii ale timpului de execuție (încadrate în dreptunghiuri pe cele două figuri) diferă, semn că la fiecare execuție s-au înregistrat alți timpi. Cauzele pot fi multiple: de la disponibilitatea sistemului de calcul până la timpul de reacție al utilizatorului la întrebări precum `GETFILE()` ori `MESSAGEBOX()`, ca în exemplul de față. Pe de altă parte, instrumentul `Coverage Profiler` poate ajuta programatorul să testeze diferite variante de codificare înaintea oferirii versiunii finale a produsului-program („ajustând" secvențele de program mari consumatoare de timp).

În fine, butonul (17) din figura 4.28 prezintă încă o opțiune foarte interesantă: `Toggle Event Logging`, cât se poate de utilă pentru depanarea formularelor. Această opțiune permite jurnalizarea apariției evenimentelor în fereastra `Debug Output` sau scrierea lor într-un fișier `.log`.

Capitolul 5

SQL în Visual FoxPro

FoxPro a fost unul dintre primele SGBD-uri xBase care au implementat un important nucleu SQL. Visual FoxPro respectă, în continuare, multe din specificaţiile standardului SQL-92, deşi nu putem vorbi de un nivel similar marilor produse ale lumii SGBDR: Oracle, DB2, Sybase, Informix, SQL Server etc. Cu toate limitele sale, care vor fi punctate pe parcursul acestui capitol, prin posibilitatea includerii comenzilor SQL în programe VFP şi folosirea cursoarelor sau tabelelor virtuale, practic în Visual FoxPro se poate rezolva orice problemă de interogare şi manipulare a datelor.

5.1. Principalele comenzi SQL din VFP

Comenzile SQL-92 implementate în Visual FoxPro 6 sunt prezentate în tabelul 6.1.

Tabelul 5.1. Comenzi SQL în Visual FoxPro

Comandă	*Scop*
Pentru manipularea datelor	
SELECT	Extragerea datelor din BD
INSERT	Adăugarea de noi linii într-o tabelă
DELETE	Ştegerea de linii dintr-o tabelă
UPDATE	Modificarea valorilor unor atribute
Pentru definirea bazei de date	
CREATE TABLE	Adăugarea unei noi tabele în BD
DROP TABLE	Ştergerea unei tabele din bază
ALTER TABLE	Modificarea structurii unei tabele
CREATE VIEW	Crearea unei tabele virtuale
DROP VIEW	Ştergerea unei tabele virtuale
CREATE CURSOR	Crearea unei tabele temporare
Pentru controlul tranzacţiilor	
BEGIN TRANSACTION	Marchează începutul unei tranzacţii
END TRANSACTION	Marchează sfârşitul unei tranzacţii
ROLLBACK	Abandonează tranzacţia în curs

Din paragraful următor vom începe prezentarea comenzilor SQL cu opţiunile esenţiale pentru crearea şi modificarea structurii unei baze date şi a tabelelor acesteia – CREATE TABLE, ALTER TABLE, inclusiv a restricţiilor ce pot fi definite la nivel de atribut sau tabelă. Pentru popularea şi actualizarea tabelelor vor fi prezentate clauzele comenzilor INSERT, UPDATE şi DELETE.

5.2. Crearea şi modificarea structurii tabelelor

Dacă în capitolul 2 au fost prezentate modalităţile asistate de creare a bazei de date, a tabelelor acesteia, inclusiv a restricţiilor, în acest capitol ne vom apropia de zona „profesională" a dezvoltării modulelor bazei de date în cadrul aplicaţiilor VFP.

5.2.1. Crearea tabelelor şi declararea restricţiilor

Comanda SQL utilizată pentru crearea unei tabele este CREATE TABLE, iar pentru modificarea ulterioară a structurii se foloseşte ALTER TABLE. Precizăm că este vorba de crearea structurii tabelei. Popularea cu înregistrări şi actualizarea ulterioară se realizează prin alte comenzi: INSERT, UPDATE, DELETE.

Formatul general al comenzii CREATE TABLE în Visual FoxPro este:

```
CREATE TABLE | DBF TableName1 [NAME LongTableName] [FREE]
      (FieldName1 FieldType [(nFieldWidth [, nPrecision])]
            [NULL | NOT NULL]
            [CHECK lExpression1 [ERROR cMessageText1]]
            [DEFAULT eExpression1]
            [PRIMARY KEY | UNIQUE]
            [REFERENCES TableName2 [TAG TagName1]]
            [NOCPTRANS]
      [, FieldName2 ...]
            [, PRIMARY KEY eExpression2 TAG TagName2
            |, UNIQUE eExpression3 TAG TagName3]
            [, FOREIGN KEY eExpression4 TAG TagName4 [NODUP]
                  REFERENCES TableName3 [TAG TagName5]]
            [, CHECK lExpression2 [ERROR cMessageText2]])
  | FROM ARRAY ArrayName
```

Iată, pe scurt, semnificaţia opţiunilor:

CREATE TABLE | DBF TableName1

TableName1 reprezintă numele tabelei ce urmează a fi creată. De remarcat că nu există nici o diferenţă între opţiunile TABLE şi DBF. Prima se adresează celor ataşaţi teoriei relaţionale, iar cea de-a doua este mai aproape de utilizatorul „tradiţional" al FoxPro (xBase-urilor, în general), pentru care o tabelă este un fişier cu extensia .DBF.

NAME LongTableName

Permite specificarea unui nume mai lung (până la 128 de caractere) pentru tabela creată. Pentru aceasta, este necesar ca baza de date să fie în prealabil deschisă, deoarece numele lungi sunt memorate în containerul asociat bazei (.DBC).

FREE

Indică faptul că tabela respectivă va fi independentă, deci nu va face parte din bază.

(FieldName1 FieldType [(nFieldWidth [, nPrecision])]

Permite declararea, pentru fiecare atribut (câmp) al tabelei, a numelui, tipului, lungimii şi, eventual, a numărului de poziţii pentru reprezentarea părţii fracţionare (zecimale).

NULL

Prin specificarea sa, atributul este autorizat să conţină valori nule (NULL).

NOT NULL

Nu permite apariţia valorilor nule pentru atributul respectiv. Automat, pentru atributele de tip cheie primară sau pentru care este utilizată opţiunea UNIQUE nu se admit valori NULL.

CHECK lExpression1

Serveşte la specificarea unei funcţii-utilizator de validare la nivel de atribut (câmp)[25]. Astfel, funcţia este verificată imediat după adăugarea unei noi înregistrări (linii) în tabelă. Dacă rezultatul evaluării funcţiei este .F. (FALSE), se declanşează o eroare.

ERROR cMessageText1

În cazul în care funcţia de validare de la nivelul atributului nu se respectă (adică „întoarce" valoarea logică FALSE), pe ecran apare mesajul cMessageText1.

DEFAULT eExpression1

Specifică valoarea implicită a atributului (în funcţie de tipul acestuia), valoare pe care o va conţine acest câmp la adăugarea unei noi înregistrări în tabelă.

PRIMARY KEY

Declară atributul respectiv cheie primară, prin crearea unui index principal cu nume identic ca al atributului.

UNIQUE

Creează un „index candidat" – altfel spus, declară acest atribut cheie alternativă. Se previne astfel apariţia a două valori identice în tabelă pentru acest câmp.

REFERENCES TableName2 [TAG TagName1]

Permite definirea unei restricţii referenţiale prin crearea unei legături permanente între tabele. TableName2 este tabela-părinte a legăturii. Dacă nu se specifică TAG TagName1, relaţia este stabilită prin intermediul indexului primar al tabelei TableName2; dacă acesta (indexul primar) nu există, se generează un mesaj de eroare. TableName2 nu poate fi o tabelă independentă.

NOCPTRANS

Este utilă pentru câmpuri de tip şir de caractere şi memo, pentru a preveni conversia la un alt cod de pagină (*code page*).

25. Vezi capitolul 2, paragraful 2.5.1.

PRIMARY KEY eExpression2 TAG TagName2

Creează automat un index primar pentru tabela curentă. eExpression2 este un câmp sau o combinaţie de câmpuri ale tabelei. Numele indexului poate conţine până la 10 caractere. Firește, clauza PRIMARY KEY nu poate apărea decât o singură dată într-o comandă CREATE TABLE.

UNIQUE eExpression3 TAG TagName3

Creează un index candidat. eExpression3 desemnează orice câmp sau combinaţie de câmpuri ale tabelei, atribut/combinaţie care va îndeplini rolul de cheie alternativă. TagName3 reprezintă numele indexului candidat care este alcătuit tot din maximum 10 caractere. Pentru o aceeaşi tabelă pot fi creaţi mai mulţi indecşi candidaţi.

FOREIGN KEY eExpression4 TAG TagName4 [NODUP]

Are ca rezultat crearea unui index obişnuit (*regular*) pentru tabela curentă (cheia de indexare fiind eExpression4, iar numele acestuia TagName4) şi stabilirea unei relaţii permanente cu o tabelă-părinte. Prin NODUP se poate crea un index străin candidat (prin analogie cu indecşii candidaţi la „postul" de index primar).

REFERENCES TableName3 [TAG TagName5]

Specifică numele tabelei-părinte implicate în legătura creată prin opţiunea FOREIGN KEY. Indexul TagName5 este cel prin care se realizează legătura dintre cele două tabele. Implicit este indexul primar al tabelei TableName3.

CHECK eExpression2 [ERROR cMessageText2]

Permite specificarea unei restricţii (reguli de validare) la nivel de tabelă; cMessageText2 este mesajul afişat în cazul nerespectării restricţiei.

FROM ARRAY ArrayName

Permite crearea unei tabele pe baza datelor conţinute într-o variabilă de tip tablou.

În listingul 5.1 este prezentat programul pentru crearea, în VFP 6, a bazei de date VINZARI.

Listing 5.1. Crearea tabelelor bazei de date în VFP 6

```
* In caz ca baza exista deja,
* pentru crearea sa, trebuie inchisa
CLOSE DATA ALL
CLOSE TABLES ALL
CLOSE ALL

* se sterge BD impreuna cu tabelele care o alcatuiesc
* Atentie !!! Comanda urmatoare distruge intreg continutul bazei !
DELETE DATABASE vinzari DELETETABLES

* A sosit vremea creatiei
CREATE DATABASE vinzari
```

```
* Secventa de creare a tabelelor

* In tabela JUDETE
*      - JUD este cheie primara
*      - literele JUD trebuie sunt exclusiv majuscule
*      - JUDET este cheie candidat (alternativa)
*      - JUDET nu poate avea valori nule
*      - prima litera din fiecare cuvint al denumirii judetului
*                este obligatoriu majuscula
*      - valoarea implicita a atributului REGIUNE este „MOLDOVA"
*      - valoarea REGIUNE trebuie sa se incadreze in lista:
*                „MOLDOVA", „Banat",....
CREATE TABLE judete ( ;
   jud CHAR(2) ;
      PRIMARY KEY ;
      CHECK (jud=LTRIM(UPPER(jud))) ;
         ERROR 'Indicativul judetului se scrie cu majuscule !', ;
   judet CHAR(25) ;
      UNIQUE ;
      NOT NULL ;
      CHECK (judet=LTRIM(PROPER(judet))) ;
         ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
         'denumirii judetului este majuscula; '+CHR(13)+;
         'restul literelor sunt mici!', ;
   regiune CHAR(15) ;
      DEFAULT 'Moldova' ;
      CHECK (INLIST(regiune, 'Banat', 'Transilvania', 'Dobrogea', 'Oltenia', ;
      'Muntenia', 'Moldova')) ;
         ERROR 'Regiunea poate avea o valoare numai din lista:'+CHR(13)+;
         'Banat, Transilvania, Dobrogea, Oltenia, Muntenia, Moldova' ;
  )


* In tabela LOCALITATI
*      - ...
*      - atributul JUD este cheie straina (parintele fiind tabela JUDETE)
CREATE TABLE localitati ( ;
   codpost CHAR(5) ;
      PRIMARY KEY ;
      CHECK (codpost=LTRIM(codpost)) ;
         ERROR 'Codul postal se scrie fara spatii la inceput !', ;
   loc CHAR(25) ;
      NOT NULL ;
      CHECK (loc=LTRIM(PROPER(loc))) ;
         ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
         'denumirii localitatii este majuscula; '+CHR(13)+;
         'restul literelor sunt mici!', ;
   jud CHAR(2) ;
      DEFAULT 'IS', ;
   FOREIGN KEY jud TAG jud REFERENCES judete TAG jud ;
   )
```



```
CREATE TABLE clienti ( ;
   codcl NUMBER(6) ;
      PRIMARY KEY ;
      CHECK (codcl > 1000), ;
   dencl CHAR(30) ;
      CHECK (SUBSTR(dencl,1,1) = UPPER(SUBSTR(dencl,1,1))) ;
         ERROR 'Prima litera din denumirea clientului este ; '+CHR(13)+;
         'obligatoriu majuscula !', ;
   codfiscal CHAR(9) ;
      NULL ;
      CHECK (SUBSTR(codfiscal,1,1) = UPPER(SUBSTR(codfiscal,1,1))) ;
         ERROR 'Prima (eventuala) litera din codul fiscal ; '+CHR(13)+;
         'este obligatoriu majuscula !', ;
   adresa CHAR(40) ;
      NULL ;
      CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
         ERROR 'Prima (eventuala) litera din adresa ; '+CHR(13)+;
         'este obligatoriu majuscula !', ;
   codpost CHAR(5), ;
   telefon CHAR(10) ;
      NULL, ;
  FOREIGN KEY codpost TAG codpost REFERENCES localitati TAG codpost ;
   ) ;


CREATE TABLE persoane ( ;
   cnp CHAR(14) ;
      PRIMARY KEY ;
      CHECK (cnp=LTRIM(UPPER(cnp))) ;
         ERROR 'Codul numeric personal se scrie fara spatii la inceput !', ;
   nume CHAR(20) ;
      CHECK (nume=LTRIM(PROPER(nume))) ;
         ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
         'numelui este majuscula; '+CHR(13)+;
         'restul literelor sunt mici!', ;
   prenume CHAR(20) ;
      CHECK (prenume=LTRIM(PROPER(prenume))) ;
         ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
         'prenumelui este majuscula; '+CHR(13)+;
         'restul literelor sunt mici!', ;
   adresa CHAR(40) ;
      NULL ;
      CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
         ERROR 'Prima litera din adresa este obligatoriu majuscula !', ;
   sex CHAR(1) DEFAULT 'B' ;
      CHECK (INLIST(sex,'F','B')) ;
         ERROR 'Atributul Sex poate avea valorile F (de la Femeiesc)'+CHR(13)+;
               'sau B (de la Barbatesc) !', ;
   codpost CHAR(5), ;
   telacasa CHAR(10) NULL, ;
   telbirou CHAR(10) NULL, ;
   telmobil CHAR(10) NULL, ;
   email CHAR(20) NULL, ;
```

```
   FOREIGN KEY codpost TAG codpost REFERENCES localitati TAG codpost ;
   ) ;


* In tabela PERSCLIENTI
*      - cheia primara este compusa: (CNP, CodCl)
CREATE TABLE persclienti ( ;
   cnp CHAR(14), ;
   codcl NUMBER(6), ;
   functie CHAR(25) ;
      CHECK (SUBSTR(functie,1,1) = UPPER(SUBSTR(functie,1,1))) ;
         ERROR 'Prima litera din functie este obligatoriu majuscula !', ;
   PRIMARY KEY cnp+STR(codcl,6)+functie TAG primaru, ;
   FOREIGN KEY cnp TAG cnp REFERENCES persoane TAG cnp, ;
   FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl ;
   ) ;

* Tabela PRODUSE
*      - contine atributul IMAGINE de tip GENERAL
CREATE TABLE produse ( ;
   codpr NUMBER(6) ;
      PRIMARY KEY ;
      CHECK (codpr > 0) ;
         ERROR 'Codul produsului trebuie sa fie mai mare de 100000 !', ;
   denpr CHAR(30) ;
      CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))) ;
      ERROR 'Prima litera din denumirea produsului este obligatoriu majuscula !', ;
   um CHAR(10), ;
   grupa CHAR(15) ;
      CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))) ;
      ERROR 'Prima litera din grupa este obligatoriu majuscula !', ;
   procTVA NUMBER(3,2) ;
      DEFAULT .22, ;
   imagine GENERAL ;
   )


CREATE TABLE gestiuni ( ;
   gestiune  CHAR(4) ;
      PRIMARY KEY, ;
   den_gest CHAR(25) ;
      CHECK (SUBSTR(den_gest,1,1) = UPPER(SUBSTR(den_gest,1,1))) ;
      ERROR 'Prima litera din denumirea gestiunii este obligatoriu majuscula !', ;
   adresa CHAR(40) ;
      NULL ;
      CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
         ERROR 'Prima litera din adresa este obligatoriu majuscula !', ;
   codpost CHAR(5), ;
   telefon CHAR(10) ;
      NULL, ;
   cnp CHAR(14), ;
   email CHAR(20) NULL, ;
```



```
   FOREIGN KEY codpost TAG codpost REFERENCES localitati TAG codpost, ;
   FOREIGN KEY cnp TAG cnp REFERENCES persoane TAG cnp ;
   )

* In tabela FACTURI
*      - valorile atributului DATAFACT trebuie sa se incadreze
*           in intervalul 1 ian.2001 - 31 dec.2010
CREATE TABLE facturi ( ;
   nrfact NUMBER(8) ;
      PRIMARY KEY, ;
   datafact DATE ;
      DEFAULT DATE() ;
      CHECK (BETWEEN(datafact,{^2001/01/01},{^2010/12/31})) ;
      ERROR 'Baza de date functioneaza in intervalul 1 ian 2001 - 31 dec.2010 !', ;
   gestiune CHAR(4) NOT NULL, ;
   codcl NUMBER(6), ;
   obs CHAR(50) NULL, ;
   FOREIGN KEY gestiune TAG gsetiune REFERENCES gestiuni TAG gestiune, ;
   FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl ;
   ) ;


CREATE TABLE liniifact ( ;
   nrfact NUMBER(8), ;
   linie NUMBER(2) ;
      CHECK (linie > 0) ;
         ERROR 'Atributul linie trebuie sa fie mai mare ca zero !', ;
   codpr NUMBER(6), ;
   cantitate NUMBER(10), ;
   pretunit NUMBER (12), ;
   PRIMARY KEY STR(nrfact,8)+STR(linie,2) TAG primaru, ;
   FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact, ;
   FOREIGN KEY codpr TAG codpr REFERENCES produse TAG codpr ;
   )


* In tabela INCASARI
*      - valorile atributelor DATAINC si DATADOC trebuie sa se incadreze
*           in intervalul 1 ian.2001 - 31 dec.2010
*      - valoarea implicita a atributului DATAINC este data curenta (data sistemului)
*      - valoarea implicita a atributului DATADOC este data curenta (data sistemului)
*      minus 5; aceasta inseamna ca, de obicei, intre emiterea documentului
*      de plata si data in care banii sosesc in cont trec 5 zile (e doar o presupunere !)
CREATE TABLE incasari ( ;
   codinc NUMBER(8) ;
      PRIMARY KEY, ;
   datainc DATE ;
      DEFAULT DATE() ;
      CHECK (BETWEEN(datainc,{^2001/01/01},{^2010/12/31})) ;
      ERROR 'Baza de date functioneaza in intervalul 1 ian.2001 - 31 dec.2010 !', ;
   coddoc CHAR(4) ;
      CHECK(coddoc=UPPER(LTRIM(coddoc))) ;
         ERROR 'Codul documentului se scrie cu majuscule !', ;
   nrdoc CHAR(16), ;
```

```
    datadoc DATE ;
       DEFAULT DATE() - 5 ;
       CHECK (BETWEEN(datadoc, {^2001/01/01},{^2001/12/31})) ;
       ERROR 'Data documentului trebuie sa fie intre 1 ian.2000 si 31 dec.2010 !' ;   ) ;

CREATE TABLE incasfact ( ;
    codinc NUMBER(8), ;
   nrfact NUMBER(8), ;
    transa NUMBER(16) ;
       NOT NULL, ;
    PRIMARY KEY STR(codinc,8)+STR(nrfact,8) TAG primaru, ;
   FOREIGN KEY codinc TAG codinc REFERENCES incasari TAG codinc, ;
   FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact ;
   ) ;
```

În VFP este obligatoriu să se indice explicit ce atribute *pot* avea valori nule, iar la declararea cheilor primare compuse, atributele componente trebuie concatenate, lucru ce atrage necesitatea conversiei celor de alt tip (numerice, dată calendaristică, logice) în şiruri de caractere. Declararea cheilor primare, alternative şi străine în VFP presupune crearea automată a indecşilor de tip `PRIMARY`, `CANDIDATE` sau `REGULAR`. Prin clauza `TAG` se poate da indexului respectiv un nume la alegere.

Din păcate, declararea cheilor străine nu înseamnă şi instituirea restricţiei referenţiale. Aceasta este una dintre cele mai ciudate „găselniţe" VFP. Ca urmare, după crearea BD, fie trebuie „umblat" prin *Referential Integrity Builder* şi, astfel, grafic, instituite regulile pentru prezervarea referenţialităţii (vezi capitolul 2), fie trebuie create declanşatoare în acest scop (capitolul 6).

5.2.2. Modificarea structurii tabelelor/restricţiilor în VFP

Schema unei baze de date reprezintă aspectul constant, invariabil sau, mai bine zis, puţin variabil în timp. O bună analiză şi proiectare a aplicaţiei (sistemului) diminuează riscul modificărilor de amploare în structura tabelelor şi restricţiilor, conferind stabilitate bazei şi diminuând sensibil eforturile de întreţinere ulterioară instalării aplicaţiei. Cu toate acestea, pe parcursul utilizării bazei de date pot apărea situaţii în care este necesară adăugarea unui nou atribut, ştergerea unui atribut inutil, modificarea lungimii etc.

Comanda pentru modificarea structurii unei tabele este `ALTER TABLE`, ale cărei opţiuni mai importante le vom parcurge în continuare.

Adăugarea unui nou atribut

Adăugarea atributului `DataNast` (data naşterii) în tabela PERSOANE se realizează astfel:

```
ALTER TABLE PERSOANE ADD DataNast DATE
```

Ştergerea unui atribut

Dacă ne propunem să ştergem atributul pe care tocmai l-am adăugat, trebuie doar să folosim comanda:

```
ALTER TABLE PERSOANE DROP COLUMN DataNast
```


Redenumirea unui atribut

Spre deosebire chiar de SGBD-uri consacrate, în VFP un atribut se poate şi redenumi:

```
ALTER TABLE PERSOANE RENAME COLUMN DataNast TO DataNasterii
```

Modificarea tipului/ lungimii unui atribut

Dacă se doreşte creşterea la 21 a numărului de caractere alocate valorilor atributului `Prenume` şi reducerea la 17 a lungimii atributului `Nume`, vom utiliza:

```
ALTER TABLE PERSOANE ALTER COLUMN Prenume CHAR(21)
ALTER TABLE PERSOANE ALTER COLUMN Nume CHAR(17)
```

Adăugarea/modificarea valorii implicite

Dacă dorim ca valoarea implicită a atributului `Sex` să fie 'F' (deoarece sunt mai multe femei decât bărbaţi), acest lucru este posibil şi astfel:

```
ALTER TABLE PERSOANE ALTER COLUMN Sex SET DEFAULT 'F'
```

NULL şi neNULL

Interzicerea valorilor nule pentru atributul `Sex` se realizează astfel:

```
ALTER TABLE PERSOANE ALTER COLUMN Sex NOT NULL
```

Invers, pentru a permite valori `NULL` pentru acelaşi atribut:

```
ALTER TABLE PERSOANE ALTER COLUMN Sex NULL
```

Adăugarea/anularea restricţiilor

Toate restricţiile: cheie primară – `PRIMARY KEY`, unicitate – `UNIQUE`, referenţială – `FOREIGN KEY`, de comportament – `CHECK` pot fi declarate ulterior creării tabelei şi, bineînţeles, anulate la un moment dat. Spre exemplu, formatul general al comenzii pentru dezactivarea cheii primare este:

```
ALTER TABLE PERSOANE DROP PRIMARY KEY
```

Pentru reinstituirea cheii primare a tabelei PERSOANE comanda are forma:

```
ALTER TABLE PERSOANE ADD PRIMARY KEY (CNP)
```

Analog, prin `ADD` şi `DROP` pot fi instituite/anulate şi celelalte tipuri de restricţii, formatele generale ale comenzii `ALTER TABLE` fiind:

```
ALTER TABLE TableName1
ADD | ALTER [COLUMN] FieldName1
FieldType [(nFieldWidth [, nPrecision])]
[NULL | NOT NULL]
[CHECK lExpression1 [ERROR cMessageText1]]
[DEFAULT eExpression1]
[PRIMARY KEY | UNIQUE]
[REFERENCES TableName2 [TAG TagName1]]
[NOCPTRANS]
[NOVALIDATE]
```

sau

```
ALTER TABLE TableName1
ALTER [COLUMN] FieldName2
[NULL | NOT NULL]
[SET DEFAULT eExpression2]
[SET CHECK lExpression2 [ERROR cMessageText2]]
[DROP DEFAULT]
[DROP CHECK]
[NOVALIDATE]
```

sau

```
ALTER TABLE TableName1
[DROP [COLUMN] FieldName3]
[SET CHECK lExpression3 [ERROR cMessageText3]]
[DROP CHECK]
[ADD PRIMARY KEY eExpression3 TAG TagName2 [FOR
lExpression4]]
[DROP PRIMARY KEY]
[ADD UNIQUE eExpression4 [TAG TagName3 [FOR lExpression5]]]
[DROP UNIQUE TAG TagName4]
[ADD FOREIGN KEY [eExpression5] TAG TagName4 [FOR
lExpression6]
REFERENCES TableName2 [TAG TagName5]]
[DROP FOREIGN KEY TAG TagName6 [SAVE]]
[RENAME COLUMN FieldName4 TO FieldName5]
[NOVALIDATE]
```

Unul dintre elementele de noutate față de formatul comenzii `CREATE TABLE` îl reprezintă apariția clauzei `NOVALIDATE`, prin prezența căreia se permite operarea unor modificări ale structurii chiar dacă acestea ar viola integritatea datelor din bază.

5.2.3. Ştergerea tabelelor

Comanda de ştergere a unei tabele din bază este `DROP TABLE` şi are o sintaxă extrem de simplă:

```
DROP TABLE tabelă
```

5.3. Comenzi de actualizare

Nucleul SQL din Visual FoxPro prezintă toate cele trei comenzi pentru actualizarea conținutului unei tabele: `INSERT`, `UPDATE` şi `DELETE`.

5.3.1. Adăugarea unei linii

Comanda SQL de adăugare de noi linii este `INSERT`, care are, în modul de lucru cel mai puțin pretențios, următorul format:


```
INSERT INTO tabelă [(atribut1, atribut2, ….)]
      VALUES (valoare_atribut1, valoare_atribut2, ….)
```

Ordinea valorilor din clauza `VALUES` trebuie să fie identică cu cea declarată la crearea (sau modificarea structurii) tabelelor. Modificarea este posibilă numai prin enumerarea, după numele tabelei, a atributelor ce vor primi valorile specificate.

Dacă, spre exemplu, pentru tabelele CLIENTI şi FACTURI au fost incluse în clauza `VALUES` mai puține valori decât atributele tabelei, VFP va genera o eroare. Este obligatorie, în aceste cazuri, precizarea atributelor care vor primi valorile (vezi listingul 5.2). În Oracle şi DB2, restul, adică cele nespecificate, vor avea, pe liniile respective, valori `NULL`. În VFP, acestea sunt completate cu valori vide (funcția `EMPTY()` întoarce `.T.`, în timp ce `ISNULL()` - `.F.`).

5.3.2. Ştergerea liniilor

Comanda SQL pentru ştergerea uneia sau mai multor linii dintr-o tabelă este `DELETE`. Formatul general este:

```
DELETE FROM nume-tabelă WHERE predicat
```

În tabelă vor fi *marcate pentru ştergere* toate liniile care îndeplinesc condiția specificată în predicatul din clauza `WHERE`. Firește, când clauza `WHERE` lipseşte, vor fi marcate pentru ştergere toate liniile tabelei.

Exemplul 1
Să se elimine din tabela FACTURI toate facturile emise de gestiunea 002.

```
DELETE FROM FACTURI ;
WHERE gestiune = '002'
```

Exemplul 2
Să se elimine din baza de date toate județele din Moldova.

```
DELETE FROM JUDETE ;
WHERE Regiune = 'Moldova'
```

5.3.3. Popularea bazei de date prin comenzi SQL

Folosind comenzile de ştergere şi inserare, în listingul 5.2 este prezentat programul pentru popularea bazei de date. La început se folosesc comenzile de ştergere, pentru a preveni eventuala duplicare a înregistrărilor.

Listing 5.2. Script Visual FoxPro de populare a tabelelor (cu valorile din capitolul 2)

```
*   Program de populare a BAZEI DE DATE
*
* se deschide baze de date (in caz ca
*   nu e deja deschisa
IF !DBUSED('vinzari')
    OPEN DATABASE vinzari EXCLUSIVE
ENDIF
```

```
DELETE FROM incasfact
SELE incasfact
ZAP

DELETE FROM incasari
SELE incasari
ZAP

DELETE FROM liniifact
SELE liniifact
ZAP

DELETE FROM facturi
SELE facturi
ZAP

DELETE FROM gestiuni
SELE gestiuni
ZAP

DELETE FROM produse
SELE produse
ZAP

DELETE FROM persclienti
SELE persclienti
ZAP

DELETE FROM persoane
SELE persoane
ZAP

DELETE FROM clienti
SELE clienti
ZAP

DELETE FROM localitati
SELE localitati
ZAP

DELETE FROM judete
SELE judete
ZAP

INSERT INTO judete VALUES ('IS', 'Iasi','Moldova')
INSERT INTO judete VALUES ('VN', 'Vrancea', 'Moldova')
INSERT INTO judete VALUES ('NT', 'Neamt', 'Moldova')
INSERT INTO judete VALUES ('SV', 'Suceava', 'Moldova')
INSERT INTO judete VALUES ('VS', 'Vaslui', 'Moldova')
INSERT INTO judete VALUES ('TM', 'Timis', 'Banat')

INSERT INTO localitati VALUES ('6600', 'Iasi', 'IS')
INSERT INTO localitati VALUES ('5725', 'Pascani', 'IS')
INSERT INTO localitati VALUES ('6500', 'Vaslui', 'VS')
INSERT INTO localitati VALUES ('5300', 'Focsani', 'VN')
INSERT INTO localitati VALUES ('6400', 'Birlad', 'VS')
INSERT INTO localitati VALUES ('5800', 'Suceava', 'SV')
INSERT INTO localitati VALUES ('5550', 'Roman', 'NT')
INSERT INTO localitati VALUES ('1900', 'Timisoara', 'TM')
```



```
INSERT INTO clienti VALUES (1001, 'Client 1 SRL', 'R1001', 'Tranzitiei, 13 bis', ;
  '6600', NULL)
INSERT INTO clienti (codcl, dencl, codfiscal, codpost, telefon) ;
  VALUES (1002,'Client 2 SA', 'R1002', '6600', '032-212121')
INSERT INTO clienti VALUES (1003, 'Client 3 SRL', 'R1003', 'Prosperitatii, 22', ;
  '6500','035-222222')
INSERT INTO clienti (codcl, dencl, adresa, codpost) ;
  VALUES (1004, 'Client 4', 'Sapientei, 56', '5725')
INSERT INTO clienti VALUES (1005, 'Client 5 SRL', 'R1005', NULL, ;
  '1900', '056-111111')
INSERT INTO clienti VALUES (1006, 'Client 6 SA', 'R1006', 'Pacientei, 33', ;
  '5550', NULL)
INSERT INTO clienti VALUES (1007, 'Client 7 SRL', 'R1007', 'Victoria Capitalismului, 2', ;
  '1900', '056-121212')

INSERT INTO persoane VALUES ('CNP1', 'Ioan', 'Vasile', 'I.L.Caragiale, 22', 'B', ;
  '6600', '123456', '987654', '094222222', NULL)
INSERT INTO persoane VALUES ('CNP2', 'Vasile', 'Ion', NULL,     'B', ;
  '6600', '234567', '876543', '094222223', 'Ion@a.ro')
INSERT INTO persoane VALUES ('CNP3', 'Popovici', 'Ioana', 'V.Micle, Bl.I, Sc.B,Ap.2', 'F', ;
  '5725', '345678', NULL, '094222224', NULL)
INSERT INTO persoane VALUES ('CNP4', 'Lazar', 'Caraion', 'M.Eminescu, 42', 'B', ;
  '6500', '456789', NULL, '094222225', NULL)
INSERT INTO persoane VALUES ('CNP5', 'Iurea', 'Simion', 'I.Creanga, 44 bis', 'B', ;
  '6500', '567890', '543210', NULL, NULL)
INSERT INTO persoane VALUES ('CNP6', 'Vasc', 'Simona', 'M.Eminescu, 13', 'F', ;
  '5725', NULL, '432109', '094222227', NULL)
INSERT INTO persoane VALUES ('CNP7', 'Popa', 'Ioanid', 'I.Ion, Bl.H2, Sc.C, Ap.45', 'B', ;
  '1900', '789012', '321098', NULL, NULL)
INSERT INTO persoane VALUES ('CNP8', 'Bogacs', 'Ildiko', 'I.V.Viteazu, 67', 'F', ;
  '5550', '890123', '210987', '094222229', NULL)
INSERT INTO persoane VALUES ('CNP9', 'Ioan', 'Vasilica', 'Garii, Bl.B4, Sc.A, Ap.1', 'F', ;
  '1900', '901234', '109876', '094222230', NULL)
INSERT INTO persoane VALUES ('CNP10', 'Tucaliuc', 'Simion', NULL, 'B', ;
  '6600', '222222', '212121', NULL, 'simion@mail.dntis.ro')
INSERT INTO persoane VALUES ('CNP11', 'Vasiliu', 'Mihai', 'Stefan cel Mare, 34', 'B', ;
  '5800', '258965', '256985', '093555555', 'mihaiv@assist.ro')
INSERT INTO persoane VALUES ('CNP12', 'Ciubotaru', 'Toader', 'Carpati, 12', 'B', ;
  '5725', '454545', '254785', NULL, NULL)

INSERT INTO persclienti VALUES ('CNP1', 1001, 'Director general')
INSERT INTO persclienti VALUES ('CNP2', 1002, 'Director general')
INSERT INTO persclienti VALUES ('CNP3', 1002, 'Sef aprovizionare')
INSERT INTO persclienti VALUES ('CNP4', 1003, 'Sef aprovizionare')
INSERT INTO persclienti VALUES ('CNP5', 1003, 'Director financiar')
INSERT INTO persclienti VALUES ('CNP6', 1004, 'Director general')
INSERT INTO persclienti VALUES ('CNP7', 1005, 'Sef aprovizionare')
INSERT INTO persclienti VALUES ('CNP8', 1006, 'Director financiar')
INSERT INTO persclienti VALUES ('CNP9', 1007, 'Sef aprovizionare')

* cimpurile GENERAL nu accepta valori NULL
INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES (1, 'Produs 1','buc', 'Tigari', .19)
INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES (2, 'Produs 2','kg', 'Bere', 0. 19)
INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES (3, 'Produs 3','kg', 'Bere', 0. 19)
INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES (4, 'Produs 4','l', 'Dulciuri', . 19)
INSERT INTO produse (codpr, denpr, um, grupa, proctva) VALUES ;
    (5, 'Produs 5','buc', 'Tigari', . 19)
```

```
* populatea cu poze. Din pacate nu exista o comanda SQL in acest sens
SELECT produse
GO TOP
nNrFisiereTIF = ADIR(aTIFuri, „*.tif")
IF nNrFisiereTIF > 0
  FOR i = 1 to nNrFisiereTIF
    APPEND GENERAL Imagine FROM aTIFuri(i,1)
    SKIP
  ENDFOR
ENDIF


INSERT INTO gestiuni VALUES ('001', 'Depozit Pacurari','Sos. Pacurari, nr 145 bis', '6600', ;
'CNP10', '212121', 'x.gest001@mail.dntis.ro')
INSERT INTO gestiuni VALUES ('002', 'Depozit Suceava','Str. Lunetistului  nr.33', '5800', ;
'CNP11', '256985', 'x.gest002@mail.dntis.ro')
INSERT INTO gestiuni VALUES ('003', 'Sectie Pascani','Bd. Trompetistului, 56', '5725', ;
'CNP12', '254785', 'x.gest003@mail.dntis.ro')


INSERT INTO facturi (nrfact, datafact, gestiune, codcl) VALUES (1111, {^2001/08/01}, '001', 1001)
INSERT INTO facturi VALUES (1112, {^2001/08/01}, '001', 1005, 'Probleme cu transportul')
INSERT INTO facturi (nrfact, datafact, gestiune, codcl) VALUES (1113, {^2001/08/01}, '001',1002)
INSERT INTO facturi (nrfact, datafact, gestiune, codcl) VALUES (1114, {^2001/08/01}, '002', 1006)
INSERT INTO facturi (nrfact, datafact, gestiune, codcl) VALUES (1115, {^2001/08/02},'002', 1001)
INSERT INTO facturi VALUES (1116, {^2001/08/02}, '001', 1007, ;
    'Pretul propus initial a fost modificat')
INSERT INTO facturi VALUES (1117, {^2001/08/03}, '002', 1001, NULL)
INSERT INTO facturi VALUES (1118, {^2001/08/04}, '003', 1001, NULL)
INSERT INTO facturi VALUES (1119, {^2001/08/07}, '003', 1003, NULL)
INSERT INTO facturi VALUES (1120, {^2001/08/07}, '003', 1001, NULL)
INSERT INTO facturi VALUES (1121, {^2001/08/07}, '001', 1004, NULL)
INSERT INTO facturi VALUES (1122, {^2001/08/07}, '003', 1005, NULL)


INSERT INTO liniifact VALUES (1111, 1, 1, 50, 10000)
INSERT INTO liniifact VALUES (1111, 2, 2, 75, 10500)
INSERT INTO liniifact VALUES (1111, 3, 5, 500, 6500)
INSERT INTO liniifact VALUES (1112, 1, 2, 80, 10300)
INSERT INTO liniifact VALUES (1112, 2, 3, 40, 7500)
INSERT INTO liniifact VALUES (1113, 1, 2, 100, 9750)
INSERT INTO liniifact VALUES (1114, 1, 2, 70, 10700)
INSERT INTO liniifact VALUES (1114, 2, 4, 30, 15800)
INSERT INTO liniifact VALUES (1114, 3, 5, 700, 6400)
INSERT INTO liniifact VALUES (1115, 1, 2, 150, 9250)
INSERT INTO liniifact VALUES (1116, 1, 2, 125, 9300)
INSERT INTO liniifact VALUES (1117, 1, 2, 100, 10000)
INSERT INTO liniifact VALUES (1117, 2, 1, 100, 9500)
INSERT INTO liniifact VALUES (1118, 1, 2, 30, 11000)
INSERT INTO liniifact VALUES (1118, 2, 1, 150, 9300)
INSERT INTO liniifact VALUES (1119, 1, 2, 35, 10900)
INSERT INTO liniifact VALUES (1119, 2, 3, 40, 7000)
INSERT INTO liniifact VALUES (1119, 3, 4, 50, 14000)
INSERT INTO liniifact VALUES (1119, 4, 5, 750, 6300)
INSERT INTO liniifact VALUES (1120, 1, 2, 80, 11200)
INSERT INTO liniifact VALUES (1121, 1, 5, 550, 6400)
INSERT INTO liniifact VALUES (1121, 2, 2, 100, 10500)
```



```
INSERT INTO incasari VALUES (1234, {^2001/08/15}, 'OP', '111', {^2000/08/10})
INSERT INTO incasari VALUES (1235, {^2001/08/15}, 'CHIT', '222', {^2000/08/15})
INSERT INTO incasari VALUES (1236, {^2001/08/16}, 'OP', '333', {^2000/08/09})
INSERT INTO incasari VALUES (1237, {^2001/08/17}, 'CEC', '444', {^2000/08/10})
INSERT INTO incasari VALUES (1238, {^2001/08/17}, 'OP', '555', {^2000/08/10})
INSERT INTO incasari VALUES (1239, {^2001/08/18}, 'OP', '666', {^2000/08/11})


INSERT INTO incasfact VALUES (1234, 1111, 5399625)
INSERT INTO incasfact VALUES (1234, 1118, 1026375)
INSERT INTO incasfact VALUES (1235, 1112, 487705)
INSERT INTO incasfact VALUES (1236, 1117, 975410)
INSERT INTO incasfact VALUES (1236, 1118, 1026375)
INSERT INTO incasfact VALUES (1236, 1120, 731557)
INSERT INTO incasfact VALUES (1237, 1117, 975410)
INSERT INTO incasfact VALUES (1238, 1113, 1160250)
INSERT INTO incasfact VALUES (1239, 1117, 369680)
```

Am pornit de la presupunerea că în directorul curent există câte un fişier grafic, cu extensia `.TIF` pentru fiecare produs, iar ordinea acestora coincinde cu ordinea produselor din tabelă, ceea ce e destul de riscant. Astfel încât procedura din listingul 5.3 este mai indicată.

Listing 5.3. Noua secvenţă pentru populatea cu imagini a câmpurilor de tip `General`

```
SELECT produse
SCAN
  nume_ = 'figura_'+LTRIM(STR(codpr,6))+'.TIF'
  IF FILE((nume_))
     APPEND GENERAL Imagine FROM (nume_) LINK
  ENDIF
ENDSCAN
```

5.3.4. Modificarea valorilor unor atribute

Pentru a modifica valoarea unuia sau mai multor atribute pe una sau mai multe linii dintr-o tabelă se foloseşte comanda `UPDATE` cu formatul general (simplificat):

```
UPDATE tabelă ;
SET atribut1 = expresie1 [, atribut2= expresie2 ….]  ;
WHERE predicat
```

Modificarea se va produce pe toate liniile tabelei care îndeplinesc condiţia formulată prin predicat.

Exemplul 3

Noul număr de telefon al clientului ce are codul 1001 este 032-313131. Să se opereze modificarea în baza de date.

Se actualizează atributul `Telefon` din tabela CLIENTI:

```
UPDATE CLIENTI ;
SET Telefon  = '032-313131' ;
WHERE CodCl = 1001
```

Exemplul 4
În cadrul unei noi relaxări fiscale, se decide creşterea procentului TVA de la 19% la 22% pentru toate produsele.

Atributul modificat este `ProcTVA` din tabela PRODUSE, pe toate liniile:

```
UPDATE PRODUSE ;
SET ProcTVA = .22 ;
```

5.4. Interogarea bazelor de date – fraza SELECT

Baza de date prezentată pe parcursul capitolului 2 este doar puţin diferită de cea din cartea *SQL. Dialecte DB2, Oracle şi Visual FoxPro* (Polirom, Iaşi, 2001), aşa încât nu mai reluăm decât câteva exemple.

5.4.1. Sintaxa de bază

Comanda `SELECT` respectă în VFP sintaxa de bază din standardul SQL-92, cele trei clauze principale fiind `SELECT`, `FROM` şi `WHERE`, la care se adaugă `ORDER BY`, `GROUP BY`, `HAVING` şi alte elemente legate de subconsultări. De asemenea, VFP foloseşte notaţia SQL-92 în materie de reprezentare a joncţiunii interne (`INNER JOIN`) şi externe (`OUTER JOIN`). Câteva exemple sunt grăitoare în acest sens.

Exemplul 1
Presupunând că plata se face la 14 zile, care sunt facturile ce urmează a fi încasate în perioada 15-18 august 2001 ?

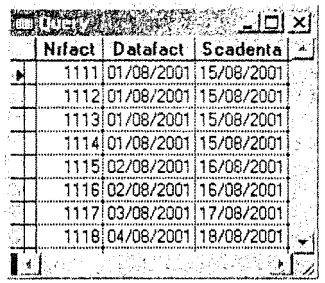
- Soluţia 1.1:

```
SELECT DISTINCT NrFact, DataFact, DataFact + 14 AS Scadenta ;
FROM FACTURI ;
WHERE DataFact + 14 >= {^2001/08/15} AND ;
  DataFact + 14 <= {^2001/08/18}
```

- Soluţia 1.2 – folosind operatorul `BETWEEN`:

```
SELECT DISTINCT NrFact, DataFact, DataFact + 14 AS Scadenta ;
FROM FACTURI ;
WHERE DataFact + 14 BETWEEN {^2001/08/15} AND {^2001/08/18}
```

Indiferent de variantă, rezultatul este cel din figura 5.1.



	Nrfact	Datafact	Scadenta
▶	1111	01/08/2001	15/08/2001
	1112	01/08/2001	15/08/2001
	1113	01/08/2001	15/08/2001
	1114	01/08/2001	15/08/2001
	1115	02/08/2001	16/08/2001
	1116	02/08/2001	16/08/2001
	1117	03/08/2001	17/08/2001
	1118	04/08/2001	18/08/2001

Figura 5.1. Facturile ce urmează a fi încasate în perioada 15-18 august 2001

Exemplul 2
Care sunt clienţii din oraşele Iaşi şi Paşcani ?

- Soluţia 2.1:

```
SELECT CLIENTI.* ;
FROM CLIENTI INNER JOIN LOCALITATI  ;
     ON CLIENTI.CodPost = LOCALITATI.CodPost ;
WHERE Loc="Iasi" OR Loc="Pascani"
```

- Soluţia 2.2 – folosind operatorul `IN`:

```
SELECT CLIENTI.* ;
FROM CLIENTI INNER JOIN LOCALITATI  ;
     ON CLIENTI.CodPost = LOCALITATI.CodPost ;
WHERE Loc IN ('Iasi', 'Pascani')
```

- Soluţia 2.3 – vechea sintaxă din SQL-89:

```
SELECT CLIENTI.* ;
FROM CLIENTI, LOCALITATI  ;
WHERE CLIENTI.CodPost = LOCALITATI.CodPost AND ;
     Loc IN ('Iasi', 'Pascani')
```

- Soluţia 2.4 – sintaxa SQL-89, cu aliasuri pentru tabele:

```
SELECT C.* ;
FROM CLIENTI C, LOCALITATI L ;
WHERE C.CodPost = L.CodPost AND ;
  Loc IN ('Iasi', 'Pascani')
```

- Soluţia 2.5 – bazată pe subconsultări:

```
SELECT * ;
FROM CLIENTI ;
WHERE CodPost IN  ;
  (SELECT CodPost ;
   FROM LOCALITATI  ;
   WHERE Loc IN ('Iasi', 'Pascani') ;
   )
```

Exemplul 3

Ce persoane-cheie de la firmele-client trebuie felicitate de Sf. Ion ?

- Soluția 3.1 – folosind operatorul LIKE:

```
SELECT dencl, functie, p.* ;
FROM PERSOANE p INNER JOIN PERSCLIENTI pc ON p.CNP=pc.CNP ;
   INNER JOIN CLIENTI c ON pc.CodCl=c.codcl ;
WHERE UPPER(Prenume) LIKE 'ION%' OR ;
   UPPER(Prenume) LIKE 'IOAN%' OR ;
   UPPER(Prenume) LIKE '% ION%' OR ;
   UPPER(Prenume) LIKE '% IOAN%' OR ;
   UPPER(Prenume) LIKE '%-ION%' OR ;
   UPPER(Prenume) LIKE '%-IOAN%'
```

	Dencl	Functie	Cnp	Nume	Prenume	Adresa	Sex	Codpost	Telacasa	Telbirou	Telmobil	Email
▶	Client 2 SA	Director general	CNP2	Vasile	Ion	NULL	B	6600	234567	876543	094222223	Ion@a.ro
	Client 2 SA	Sef aprovizionare	CNP3	Popovici	Ioana	V.Micle, BLI, Sc.B,Ap.2	F	5725	345678	NULL	094222224	NULL
	Client 5 SRL	Sef aprovizionare	CNP7	Popa	Ioanid	I.Ion, BLH2, Sc.C, Ap.45	B	1900	789012	321098	NULL	NULL

Figura 5.2. VIP-urile de la firmele-client pe care trebuie să le salutăm de Sf. Ion

Exemplul 4

Ce persoane-cheie de la firmele-client nu au cont de e-mail ?

- Soluția 4.1 – folosind operatorul IS NULL:

```
SELECT dencl, functie, p.* ;
FROM PERSOANE p INNER JOIN PERSCLIENTI pc ON p.CNP=pc.CNP ;
      INNER JOIN CLIENTI c ON pc.CodCl=c.codcl ;
WHERE email IS NULL
```

Exemplul 5

Care sunt facturile emise în aceeași zi ca și factura 1113 ?

- Soluția 5.1 – bazată pe subconsultare:

```
SELECT * ;
FROM FACTURI ;
WHERE DataFact IN ;
   (SELECT DataFact ;
    FROM FACTURI ;
    WHERE NrFact=1113)
```

- Soluția 5.2 – bazată pe joncțiune:

```
SELECT F1.NrFact, F1.DataFact  ;
FROM FACTURI F1 INNER JOIN FACTURI F2  ;
   ON F1.DataFact = F2.DataFact AND F2.NrFact = 1113
```


- Soluția 5.3 – folosind o interogare corelată:

```
SELECT * ;
FROM FACTURI F1 ;
WHERE EXISTS ;
   (SELECT 1 ;
    FROM FACTURI F2 ;
    WHERE F1.DataFact = F2.DataFact AND F2.NrFact=1113)
```

Exemplul 6

Care dintre persoanele din tabela cu același nume nu au funcții la firmele-client ?

- Soluția 6.1 – pe bază de subconsultare:

```
SELECT p.* ;
FROM PERSOANE p ;
WHERE CNP NOT IN ;
   (SELECT CNP ;
     FROM PERSCLIENTI)
```

- Soluția 6.2 – bazată pe joncțiune externă:

```
SELECT * ;
FROM PERSOANE p LEFT OUTER JOIN PERSCLIENTI pc ;
      ON p.CNP=pc.CNP ;
WHERE NVL(pc.CodCl,0) = 0
```

5.4.2. Funcții-agregat, cu și fără grupare

Exemplul 7

La câți clienți s-au trimis facturi ?

- Soluția 7.1 – o funcție COUNT și o subconsultare:

```
SELECT COUNT (*) ;
FROM CLIENTI ;
WHERE CodCl IN  ;
      (SELECT CodCl  ;
       FROM FACTURI)
```

- Soluția 7.2 – clauza DISTINCT a unei funcții COUNT:

```
SELECT COUNT (DISTINCT CodCl) ;
FROM FACTURI
```

Exemplul 8

Care este valoarea totală a facturii 1119, valoarea medie, maximă și minimă a produselor vândute pe această factură ?

Scopul exemplului este de a ilustra modul de folosirea a funcțiilor-agregat SUM, AVG, MIN, MAX. Rezultatul este prrezentat în figura 5.3.

```
SELECT SUM(Cantitate * PretUnit * ( 1 + ProcTVA) ;
      AS Val_Totala, ;
   AVG(Cantitate * PretUnit + Cantitate * PretUnit * ProcTVA);
```

```
    AS Val_Medie, ;
  MIN(Cantitate * PretUnit + Cantitate * PretUnit * ProcTVA);
    AS Val_Minima, ;
  MAX(Cantitate * PretUnit + Cantitate * PretUnit * ProcTVA);
     AS Val_Maxima ;
FROM PRODUSE P INNER JOIN LINIIFACT LF ON P.CodPr = LF.CodPr
```

Val_totala	Val_medie	Val_minima	Val_maxima
36742740.00	1670124.55	341600.00	5764500.00

Figura 5.3. Valorile: totală, medie, minimă şi maximă ale facturii 1119

Exemplul 9

Care este valoarea totală a vânzărilor pentru fiecare zi în care s-au emis facturi ?

Pentru rezolvarea problemei este obligatorie folosirea grupării (figura 5.4).

```
SELECT  DataFact, SUM(Cantitate * PretUnit * (1+ProcTVA));
    AS ValTotala ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F  ;
WHERE LF.CodPr = P.CodPr AND LF.NrFact = F.NrFact ;
GROUP BY DataFact
```

Datafact	Valtotala
01/08/2001	15054190.00
02/08/2001	3111000.00
03/08/2001	2379000.00
04/08/2001	2104500.00
07/08/2001	14094050.00

Figura 5.4. Folosirea clauzei GROUP BY

Exemplul 10

Care sunt vânzările, cantitativ şi valoric, pentru fiecare produs ?

```
SELECT DenPr, UM, SUM(Cantitate) AS Cantitativ, ;
SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Valoric ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F ;
WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
GROUP BY DenPr, UM
```

Exemplul 11

Să se obţină situaţia vânzărilor pe clienţi şi zile, afişându-se câte un subtotal la nivel de client şi un total general.

Varianta propusă reuneşte mulţimea facturilor propriu-zise, obţinută din prima frază SELECT, cu mulţimea subtotalurilor la nivel de client (al doilea SELECT) şi cu o linie ce reprezintă totalul general.


```
SELECT PADR(ALLT(DenCl),40) AS DenumireClient,  ;
  DataFact, ;
  SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C ;
WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
    AND F.CodCl=C.CodCl ;
GROUP BY DenCl, DataFact ;
  UNION  ;
    SELECT PADR(ALLT(DenCl)+'-Subtotal',40), ;
           {  /  /  }, ;
           SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
    FROM  LINIIFACT LF, PRODUSE P, FACTURI F, CLIENTI C ;
    WHERE P.CodPr = LF.CodPr AND LF.NrFact = F.NrFact ;
           ND F.CodCl=C.CodCl ;
    GROUP BY DenCl ;
  UNION  ;
SELECT  CHR(255)+'TOTAL GENERAL' , ;
  {  /  /  }, ;
  SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
FROM  LINIIFACT LF, PRODUSE P ;
WHERE P.CodPr = LF.CodPr
```

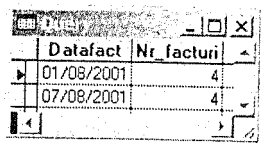
Denumireclient	Datafact	Vinzari
Client 1 SRL	01/08/2001	5535750.00
Client 1 SRL	02/08/2001	1692750.00
Client 1 SRL	03/08/2001	2379000.00
Client 1 SRL	04/08/2001	2104500.00
Client 1 SRL	07/08/2001	1093120.00
Client 1 SRL-Subtotal	/ /	12805120.00
Client 2 SA	01/08/2001	1189500.00
Client 2 SA-Subtotal	/ /	1189500.00
Client 3 SRL	07/08/2001	7425530.00
Client 3 SRL-Subtotal	/ /	7425530.00
Client 4	07/08/2001	5575400.00
Client 4-Subtotal	/ /	5575400.00
Client 5 SRL	01/08/2001	1371280.00
Client 5 SRL-Subtotal	/ /	1371280.00
Client 6 SA	01/08/2001	6957660.00
Client 6 SA-Subtotal	/ /	6957660.00
Client 7 SRL	02/08/2001	1418250.00
Client 7 SRL-Subtotal	/ /	1418250.00
ÿTOTAL GENERAL	/ /	36742740.00

Figura 5.5. Mini-OLAP în SQL-VFP

Exemplul 12
Care sunt zilele în care s-au întocmit cel puţin trei facturi ?

```
SELECT DataFact, COUNT(*) AS Nr_Facturi ;
FROM FACTURI ;
GROUP BY DataFact ;
HAVING COUNT(*) >= 3
```

Este o primă situaţie în care se recurge la opţiunea HAVING.



Datafact	Nr_facturi
01/08/2001	4
07/08/2001	4

Figura 5.6. Zilele în care s-au întocmit minimum 3 facturi

Exemplul 13
În ce zile s-au vândut şi produsul cu denumirea „Produs 1" şi cel cu denumirea „Produs 2" ?

Deşi are un enunţ banal, rezolvarea acestei probleme oferă oportunitatea unor interogări dintre cele mai spumoase. Vă prezentăm numai cinci dintre acestea.

- Soluţia 13.1 – joncţionarea unei instanţe obţinută prin joncţiunea PRODUSE-LINIIFACT-FACTURI (în care DenPr = 'Produs 1') cu o altă instanţă a aceleiaşi combinaţii (în care DenPr = 'Produs 2'):

```
SELECT DISTINCT F1.DataFact ;
FROM PRODUSE P1 ;
    INNER JOIN LINIIFACT LF1 ON P1.CodPr = LF1.CodPr ;
    INNER JOIN FACTURI F1 ON LF1.NrFact = F1.NrFact  ;
    INNER JOIN FACTURI F2 ON F1.DataFact=F2.DataFact ;
    INNER JOIN LINIIFACT LF2 ON LF2.NrFact = F2.NrFact ;
    INNER JOIN PRODUSE P2 ON LF2.CodPr = P2.CodPr  ;
WHERE P1.DenPr = 'Produs 1' AND P2.DenPr = 'Produs 2'
```

- Soluţia 13.2 – folosind clauza HAVING şi funcţia COUNT:

```
SELECT DISTINCT DataFact ;
FROM PRODUSE INNER JOIN LINIIFACT ;
     ON PRODUSE.CodPr = LINIIFACT.CodPr ;
  INNER JOIN FACTURI ON LINIIFACT.Nrfact =FACTURI.NrFact ;
WHERE DenPr IN ('Produs 1', 'Produs 2') ;
GROUP BY DataFact ;
HAVING COUNT(DISTINCT LINIIFACT.CodPr) = 2
```


- Soluţia 13.3 – simulând intersecţia prin subconsultări:

```
SELECT DISTINCT DataFact ;
FROM PRODUSE INNER JOIN LINIIFACT ;
     ON PRODUSE.CodPr = LINIIFACT.CodPr ;
  INNER JOIN FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact ;
WHERE  DenPr = 'Produs 1' AND DataFact IN ;
  (SELECT DISTINCT DataFact ;
   FROM PRODUSE ;
      INNER JOIN LINIIFACT ON ;
            PRODUSE.CodPr = LINIIFACT.CodPr ;
      INNER JOIN FACTURI ON ;
            LINIIFACT.Nrfact = FACTURI.NrFact ;
   WHERE DenPr = 'Produs 2')
```

- Soluţia 13.4 – prin corelarea a două instanţe ale joncţiunii PRODUSE-LINIIFACT-FACTURI; prima conţine liniile legate de *Produs 1*, iar a doua de *Produs 2*:

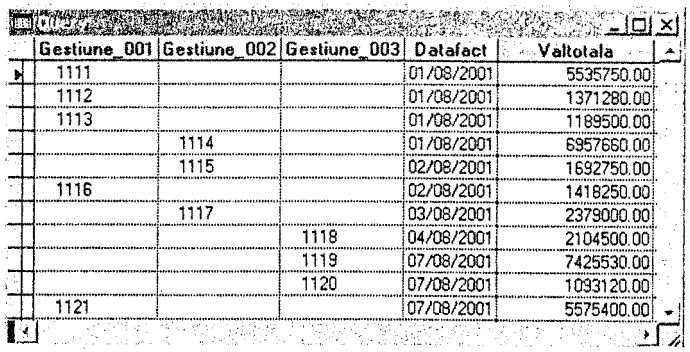
```
SELECT DISTINCT DataFact ;
FROM PRODUSE P1, LINIIFACT LF1, FACTURI F1 ;
WHERE P1.CodPr = LF1.CodPr AND ;
     LF1.Nrfact = F1.NrFact AND DenPr = 'Produs 1' ;
  AND EXISTS ;
  (SELECT 1 ;
   FROM PRODUSE P2, LINIIFACT LF2, FACTURI F2 ;
   WHERE P2.CodPr = LF2.CodPr AND LF2.Nrfact = F2.NrFact ;
     AND P2.DenPr = 'Produs 2'  AND F2.DataFact=F1.DataFact)
```

- Soluţia 13.5:

```
SELECT DISTINCT DataFact ;
FROM FACTURI  ;
  INNER JOIN LINIIFACT ON FACTURI.NrFact=LINIIFACT.NrFact ;
  INNER JOIN PRODUSE ON LINIIFACT.CodPr=PRODUSE.CodPr AND ;
           DenPr IN ('Produs 1','Produs 2') ;
WHERE DTOC(DataFact)+DTOC({//}) NOT IN ;
  (SELECT DISTINCT ;
     DTOC(F1.DataFact)+DTOC(NVL(F2.DataFact,{//})) ;
   FROM FACTURI F1 INNER JOIN PRODUSE P1 ;
           ON P1.DenPr IN ('Produs 1','Produs 2') ;
     LEFT OUTER JOIN (LINIIFACT LF2 INNER JOIN FACTURI F2 ;
           ON LF2.NrFact=F2.NrFact) ;
     ON F1.DataFact=F2.DataFact AND P1.CodPr=LF2.CodPr)
```

Exemplul 14

Să se afişeze o situaţie a facturilor emise, pe gestiuni, de forma celei din figura 5.7.



Gestiune_001	Gestiune_002	Gestiune_003	Datafact	Valtotala
1111			01/08/2001	5535750.00
1112			01/08/2001	1371280.00
1113			01/08/2001	1189500.00
	1114		01/08/2001	6957660.00
	1115		02/08/2001	1692750.00
1116			02/08/2001	1418250.00
	1117		03/08/2001	2379000.00
		1118	04/08/2001	2104500.00
		1119	07/08/2001	7425530.00
		1120	07/08/2001	1093120.00
1121			07/08/2001	5575400.00

Figura 5.7. Folosirea IIF-urilor în interogări

Afişarea pe coloane separate, pentru cele trei gestiuni, a facturilor necesită folosirea unei secvenţe de IF-uri imediate (IIF-uri):

```
SELECT IIF(gestiune = '001', STR(F.NrFact,8), space(8)) ;
      AS Gestiune_001, ;
   IIF(gestiune = '002', STR(F.NrFact,8), space(8)) ;
      AS Gestiune_002, ;
   IIF(gestiune = '003', STR(F.NrFact,8), space(8)) ;
      AS Gestiune_003, ;
   DataFact, SUM(Cantitate * PretUnit * (1+ProcTVA)) ;
      AS ValTotala ;
FROM FACTURI F ;
   INNER JOIN LINIIFACT LF ON F.NrFact = LF.NrFact ;
   INNER JOIN PRODUSE P ON LF.CodPr = P.CodPr ;
GROUP BY F.NrFact
```

Exemplul 15

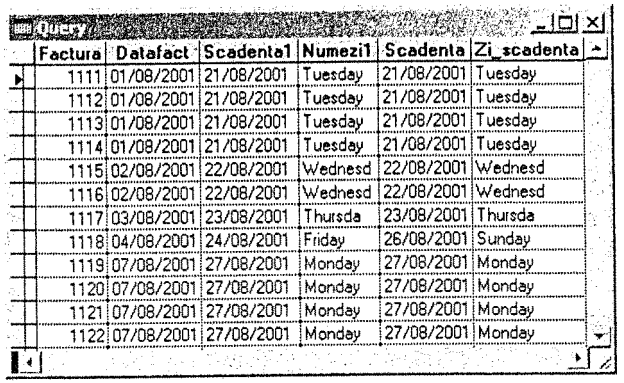
Scadenţa fiecărei facturi emise este de 20 de zile. Dacă însă data-limită cade într-o sâmbătă sau duminică, atunci scadenţa se mută în lunea următoare. Care sunt zilele scadente în aceste condiţii ?

Visual FoxPro prezintă funcţiile CDOW (*Character Day Of the Week*) şi DOW (*Day Of the Week*), rezultatul din figura 5.8 fiind obţinut după cum urmează:

```
SELECT NrFact AS Factura, DataFact, ;
   DataFact + 20  AS Scadenta1, ;
   CDOW(DataFact+20)  AS NumeZi1, ;
   IIF(DOW(DataFact+20)=6, ;
       DataFact+22, ;
       IIF(DOW(DataFact+20)=1, ;
             DataFact+21, ;
             DataFact+20) ) AS Scadenta, ;
```



```
   CDOW(IIF(DOW(DataFact+20)=6, ;
       DataFact+22, ;
       IIF(DOW(DataFact+20)=1, ;
             DataFact+21, ;
             DataFact+20) ) ) AS Zi_Scadenta ;
FROM FACTURI
```



Factura	Datafact	Scadenta1	Numezi1	Scadenta	Zi_scadenta
1111	01/08/2001	21/08/2001	Tuesday	21/08/2001	Tuesday
1112	01/08/2001	21/08/2001	Tuesday	21/08/2001	Tuesday
1113	01/08/2001	21/08/2001	Tuesday	21/08/2001	Tuesday
1114	01/08/2001	21/08/2001	Tuesday	21/08/2001	Tuesday
1115	02/08/2001	22/08/2001	Wednesd	22/08/2001	Wednesd
1116	02/08/2001	22/08/2001	Wednesd	22/08/2001	Wednesd
1117	03/08/2001	23/08/2001	Thursda	23/08/2001	Thursda
1118	04/08/2001	24/08/2001	Friday	26/08/2001	Sunday
1119	07/08/2001	27/08/2001	Monday	27/08/2001	Monday
1120	07/08/2001	27/08/2001	Monday	27/08/2001	Monday
1121	07/08/2001	27/08/2001	Monday	27/08/2001	Monday
1122	07/08/2001	27/08/2001	Monday	27/08/2001	Monday

Figura 5.8. Determinarea scadenţei de încasare

Exemplul 16

Care sunt valorile facturate şi încasate ale fiecărei facturi ?

Soluţia de mai jos, ce obţine liniile din figura 5.9, reuneşte facturile care au măcar o tranşă de încasare cu cele neîncasate deloc:

```
SELECT F. NrFact, ;
   SUM(Cantitate * PretUnit * (1+ProcTVA)) / ;
   COUNT(DISTINCT I.CodInc) AS Facturat, ;
   SUM(Transa) / MAX(LF.Linie) AS Incasat ;
FROM  LINIIFACT LF, PRODUSE P, FACTURI F, INCASFACT I ;
WHERE LF.CodPr = P.CodPr AND  LF.NrFact = F.NrFact ;
   AND F.NrFact=I.NrFact ;
GROUP BY F.NrFact ;
   UNION ;
   SELECT F. NrFact, ;
      SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat, ;
            0 AS Incasat ;
   FROM  LINIIFACT LF, PRODUSE P, FACTURI F ;
   WHERE LF.CodPr = P.CodPr AND  LF.NrFact = F.NrFact ;
      AND F.NrFact NOT IN ;
            (SELECT NrFact ;
             FROM INCASFACT)  ;
   GROUP BY F.NrFact
```

Nrfact	Facturat	Incasat
1111	5535750.0000	5399625.0000
1112	1371280.0000	487705.0000
1113	1189500.0000	1160250.0000
1114	6957660.0000	0.0000
1115	1692750.0000	0.0000
1116	1418250.0000	0.0000
1117	2379000.0000	2320500.0000
1118	2104500.0000	2052750.0000
1119	7425530.0000	0.0000
1120	1093120.0000	731557.0000
1121	5575400.0000	0.0000

Figura 5.9. Valorile facturate şi încasate ale fiecărei facturi

5.4.3. Clauza TOP. Operatorii ALL şi ANY (SOME)

Exemplul 17

Care sunt cele mai mari cinci preţuri unitare de vânzare, produsele şi facturile în care apar cele cinci preţuri maxime?

În alte SGBD-uri, această problemă ar crea dureri de cap plenare. Clauza `TOP` din VFP realizează ordonarea şi extragerea celor cinci valori – vezi figura 5.10.

```
SELECT TOP 5 NrFact, DenPr, PretUnit ;
FROM LINIIFACT INNER JOIN PRODUSE ;
      ON LINIIFACT.CodPr=PRODUSE.CodPr  ;
ORDER BY PretUnit DESC
```

Nrfact	Denpr	Pretunit
1114	Produs 4	15800
1119	Produs 4	14000
1120	Produs 2	11200
1118	Produs 2	11000
1119	Produs 2	10900

Figura 5.10. Folosirea clauzei `TOP`

Exemplul 18

Care sunt produsele vândute la preţuri unitare superioare oricărui preţ unitar la care a fost vândut 'Produs 1' ?

Unul dintre obiectivele acestui exemplu este de a prezenta modul de conexiune a unei consultări cu subconsultarea sa prin operatorul `ALL`.


```
SELECT DISTINCT DenPr, PretUnit ;
FROM PRODUSE P, LINIIFACT LF ;
WHERE P.CodPr=LF.CodPr AND PretUnit > ALL ;
      (SELECT DISTINCT PretUnit ;
       FROM PRODUSE P, LINIIFACT LF ;
       WHERE P.CodPr=LF.CodPr AND DenPr ='Produs 1')
```

Denpr	Pretunit
Produs 2	10300
Produs 2	10500
Produs 2	10700
Produs 2	10900
Produs 2	11000
Produs 2	11200
Produs 4	14000
Produs 4	15800

Figura 5.11. Operatorul `ALL`

Exemplul 19

Care sunt produsele vândute la preţuri unitare superioare ***măcar*** *unui preţ unitar al 'Produsului 1' ?*

Este genul de situaţii în care se foloseşte `SOME` sau `ANY` (au aceeaşi funcţiune).

```
SELECT DISTINCT DenPr, PretUnit ;
FROM PRODUSE P INNER JOIN LINIIFACT LF ;
  ON P.CodPr=LF.CodPr ;
WHERE PretUnit > ANY ;
      (SELECT DISTINCT PretUnit ;
       FROM PRODUSE P INNER JOIN  LINIIFACT LF ;
            ON P.CodPr=LF.CodPr ;
       WHERE DenPr ='Produs 1')
```

Denpr	Pretunit
Produs 1	9500
Produs 1	10000
Produs 2	9750
Produs 2	10000
Produs 2	10300
Produs 2	10500
Produs 2	10700
Produs 2	10900
Produs 2	11000
Produs 2	11200
Produs 4	14000
Produs 4	15800

Figura 5.12. Operatorul `ANY`

Exemplul 20

Care este ultima factură întocmită (factura cea mai recentă) şi data în care a fost emisă ?

Şi aceasta este o problemă cu multe variante de rezolvare:

- Soluţia 20.1:

```
SELECT DataFact, NrFact AS UltimaFactura ;
FROM FACTURI ;
WHERE NrFact IN  ;
   (SELECT MAX(NrFact) ;
    FROM FACTURI)
```

- Soluţia 20.2 – în locul operatorului IN, operatorul de conexiune este semnul egal:

```
SELECT DataFact, NrFact AS UltimaFactura ;
FROM FACTURI ;
WHERE NrFact = ;
   (SELECT MAX(NrFact) ;
    FROM FACTURI)
```

- Soluţia 20.3 – folosind operatorul ANY:

```
SELECT DataFact, NrFact AS UltimaFactura ;
FROM FACTURI ;
WHERE NrFact =ANY ;
   (SELECT MAX(NrFact) ;
    FROM FACTURI)
```

- Soluţia 20.4 – operatorul ALL şi, în subconsultare, funcţia MAX:

```
SELECT DataFact, NrFact AS UltimaFactura ;
FROM FACTURI ;
WHERE NrFact =ALL ;
   (SELECT MAX(NrFact) ;
    FROM FACTURI)
```

- Soluţia 20.5 – conexiune prin >= ALL:

```
SELECT DataFact, NrFact AS UltimaFactura ;
FROM FACTURI ;
WHERE NrFact >= ALL  ;
   (SELECT NrFact ;
    FROM FACTURI) ;
```

- Soluţia 20.6 – soluţie „tipică" VFP – ce foloseşte clauza TOP aplicată grupurilor:

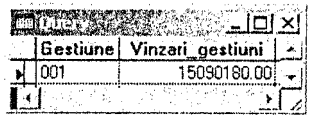
```
SELECT TOP 1 DataFact, COUNT(*) AS Nr ;
FROM FACTURI ;
GROUP BY DataFact ;
ORDER BY Nr DESC
```


Exemplul 21

Care este gestiunea cu vânzările cele mai bune ?

Dacă în alte SGBD-uri sunt necesare subconsultări în clauza FROM, în VFP, aceaşi clauză TOP este salvatoare – figura 5.13:

```
SELECT TOP 1 Gestiune,  SUM(Cantitate * PretUnit * ;
   (1+ProcTVA)) AS Vinzari_Gestiuni ;
FROM FACTURI F ;
  INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
  INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
GROUP BY Gestiune ;
ORDER BY Vinzari_Gestiuni DESC
```



Gestiune	Vinzari_gestiuni
001	15090180.00

Figura 5.13. Gestiunea pentru care s-au înregistrat cele mai mari vânzări

5.5. Actualizarea datelor folosind (sub)consultări

Visual FoxPro prezintă, de mai bine de şase ani, câteva atuuri ce constituiau altădată apanajul lumii bune a SGBD-urilor: procedurile stocate şi, implicit, declanşatoarele. Devine astfel posibilă introducerea unor atribute redundante, dar care măresc sensibil viteza de lucru a aplicaţiei. Procesul se numeşte denormalizare şi frământă lumea proiectanţilor de baze de date (nu pe noi, autorii de faţă). Aceste atribute pot fi gestionate automat şi ferite de accidente voite sau întâmplătoare.

Exagerând poate (un pic) cu denormalizarea, introducem o serie de atribute noi, după cum urmează:

- în tabela LINIIFACT – atributul ValCuTVA, pentru a elimina, la obţinerea rapoartelor/informaţiilor în rapoarte, joncţiunea cu tabela PRODUSE (din care se preia procentul de TVA);
- în tabela FACTURI adăugăm nu mai puţin de patru atribute:
  - ValTotală – reprezintă valoarea totală (inclusiv TVA) a facturii;
  - ValIncasata – este valoarea încasată (până în prezent) din contravaloarea facturii;
  - Reduceri – de obicei pentru plata rapidă a facturii;
  - Penalităţi - este opusul atributului anterior.

Comenzile SQL prin care ne punem în operă intenţiile sunt:

```
ALTER TABLE LINIIFACT ADD ValCuTVA NUMERIC(16)
ALTER TABLE FACTURI ADD ValTotala NUMERIC(16)
ALTER TABLE FACTURI ADD ValIncasata NUMERIC(16)
```

```
ALTER TABLE FACTURI ADD Reduceri NUMERIC (15)
ALTER TABLE FACTURI ADD Penalizari NUMERIC (15)
```

Deoarece aceste modificări nu au fost luate în calcul în momentul creării bazei, atributele trebuie „umplute" din mers. Motiv de prezentare a comenzilor de actualizare într-o variantă mai de finețe – folosind subconsultări. Lucrul acesta nu e permis încă de comanda INSERT, în care valorile nu pot fi specificate printr-un SELECT (spre deosebire de SQL-92).

În schimb, comanda DELETE este destul de tolerantă în acest sens. Spre exemplu, se dorește *ștergerea tuturor liniilor din tabela FACTURI care nu au copii în LINIIFACT* (altfel spus, facturile fără nici un produs/serviciu). O soluție elegantă este următoarea:

```
DELETE FROM FACTURI ;
WHERE NrFact NOT IN ;
   (SELECT DISTINCT NrFact  ;
    FROM LINIIFACT )
```

Firește, subconsultarea poate fi și corelată:

```
DELETE FROM FACTURI ;
WHERE NOT EXISTS ;
   (SELECT 1 ;
    FROM LINIIFACT ;
   WHERE LINIIFACT.NrFact = FACTURI.NrFact) ;
```

Să se elimine din tabela FACTURI toate liniile care se referă la facturi cu valoare sub 5 milioane lei.

Rezolvarea problemei presupune folosirea unei subconsultări cu grupare.

```
DELETE FROM FACTURI ;
WHERE NrFact NOT IN ;
   (SELECT NrFact ;
    FROM LINIIFACT LF INNER JOIN PRODUSE P ;
            ON LF.CodPr=P.CodPr ;
    GROUP BY NrFact ;
    HAVING SUM(Cantitate*PretUnit*(1+ProcTVA)) >= 5000000  ;
    )
```

Pentru actualizarea atributelor introduse în acest paragraf, ne interesează, în primul rând, comanda UPDATE. Dacă am fi siguri că toate produsele/serviciile ar avea procentul de TVA unic – și anume 19% –, actualizarea atributului ValCuTVA din LINIIFACT ar fi ușoară:

```
UPDATE LINIIFACT ;
   SET ValCuTVA = Cantitate * PretUnit * 1.19
```

Schema bazei este una pe termen lung însă. În funcție de legislația în vigoare, la un moment dat, produsele pot avea procente diferite de TVA, în funcție de categoria din care fac parte (de strictă necesitate, de lux etc.). De aceea, la calculul valorii cu TVA este obligatorie preluarea procentului TVA din tabela PRODUSE. Din păcate, varianta care ne-ar trebui:


```
UPDATE LINIIFACT ;
SET ValCuTVA = ;
   (SELECT Cantitate * PretUnit * (1+ProcTVA) ;
    FROM LINIIFACT LF INNER JOIN PRODUSE P ;
            ON LF.CodPr=P.CodPr ;
    WHERE LF.NrFact = LINIIFACT.NrFact AND ;
       P.CodPr = LINIIFACT.CodPr ;
    )
```

nu are de gând să funcționeze în nici una dintre versiunile VFP. Drept pentru care suntem nevoiți să folosim o soluție procedurală, cu sau fără cursoare (vezi paragraful următor). Singurul loc în care UPDATE-ul poate prezenta o subconsultare este clauza WHERE. De exemplu, dacă se dorește reducerea penalizărilor pentru toți clienții din Pașcani, se poate folosi comanda:

```
UPDATE FACTURI ;
SET Penalizari = Penalizari * .5 ;
WHERE CodCl IN ;
   (SELECT CodCl ;
    FROM CLIENTI INNER JOIN LOCALITATI ;
       ON CLIENTI.CodPost = LOCALITATI.CodPost ;
    WHERE Loc = 'Pascani')
```

5.6. Includerea frazelor SELECT în programe

Multe dintre problemele complexe de interogare necesită în SQL folosirea mai multor niveluri de subconsultare sau prezența subconsultărilor în clauzele FROM sau HAVING, facilități pe care nucleul SQL din Visual FoxPro nu le are încă implementate. În schimb, pentru aceste cazuri se pot crea programe în care rezultatele intermediare ale unor interogări sunt stocate în cursoare sau tabele virtuale care, la rândul lor, pot constitui argumente ale unor noi consultări.

5.6.1. Salvarea rezultatului unei fraze SELECT în cursor

Cursorul este, în Visual FoxPro, o tabelă temporară, a cărei viață se întinde de la momentul unui SELECT ce prezintă clauza INTO CURSOR sau CREATE CURSOR până la închiderea sa, implicită sau explicită.

Revenim la **exemplul 13**:

În ce zile s-au vândut și produsul cu denumirea „Produs 1", și cel cu denumirea „Produs 2" ?)

La seria interogărilor care au fost prezentate adăugăm și o variantă simplă de program – listingul 5.4 –, care se bazează pe folosirea a două cursoare, astfel:

Listing 5.4. Program VFP ce salvează rezultatele intermediare ale interogărilor în cursoare

```
SELECT DISTINCT DataFact ;
INTO CURSOR Zile_P1 ;
FROM PRODUSE INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr ;
  INNER JOIN FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact ;
WHERE  DenPr = 'Produs 1'

SELECT DISTINCT DataFact ;
INTO CURSOR Zile_P2 ;
FROM PRODUSE INNER JOIN LINIIFACT ON PRODUSE.CodPr = LINIIFACT.CodPr ;
  INNER JOIN FACTURI ON LINIIFACT.Nrfact = FACTURI.NrFact ;
WHERE  DenPr = 'Produs 2'

SELECT * ;
FROM Zile_P1 ;
WHERE DataFact IN  ;
  (SELECT DataFact ;
   FROM Zile_P2)
```

Prima interogare creează cursorul `Zile_P1`, ce conține numai zilele în care s-a vândut primul produs; al doilea cursor este populat cu zilele în care s-a vândut al doilea produs; ultima interogare realizează intersecția celor două cursoare – figura 5.14.

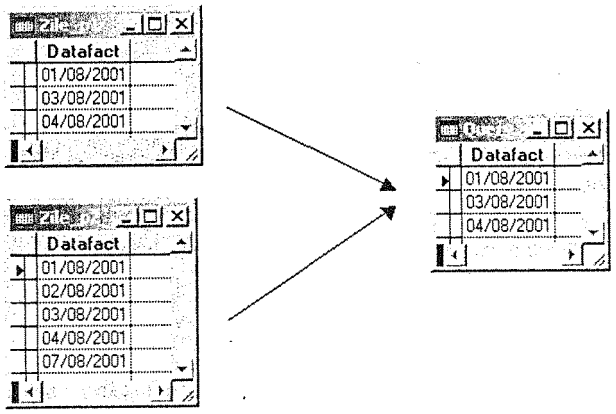


Figura 5.14. Conținutul celor două cursoare și rezultatul interogării finale

Exemplul 22

Care este județul cu vânzări imediat superioare județului Neamț ?

Soluția din listingul 5.5 are un dram suplimentar de interes, deoarece la ultima sa interogare, în afara clauzei `WHERE`, care conține două subconsultări, se efectuează theta-joncțiunea dintre cele două cursoare, `cVinzari_Neamt` și `cVinzari_Judete`.

Listing 5.5. Județul cu vânzări imediat peste cele ale județului Neamț

```
* Vinzarile jud. Neamt
SELECT SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM JUDETE J ;
    INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
    INNER JOIN CLIENTI C ON L.CodPost=C.CodPost ;
    INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
    INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
    INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cVinzari_Neamt     ;
WHERE Judet='Neamt'

* Vinzarile fiecarui judet
SELECT Judet, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Vinzari ;
FROM JUDETE J ;
    INNER JOIN LOCALITATI L ON J.Jud=L.Jud ;
    INNER JOIN CLIENTI C ON L.CodPost=C.CodPost ;
    INNER JOIN FACTURI F ON C.CodCl=F.CodCl ;
    INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
    INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cVinzari_Judete     ;
GROUP BY Judet

SELECT Judet, Vinzari ;
FROM cVinzari_Judete ;
WHERE Vinzari <= ALL ;
    (SELECT cVinzari_Judete.Vinzari ;
     FROM cVinzari_Judete, cVinzari_Neamt ;
     WHERE cVinzari_Judete.Vinzari > cVinzari_Neamt.Vinzari) ;
     AND  Vinzari > ALL ;
        (SELECT Vinzari ;
         FROM cVinzari_Neamt)
```

Conținutul cursoarelor și al rezultatului final constituie obiectul figurii 5.15.

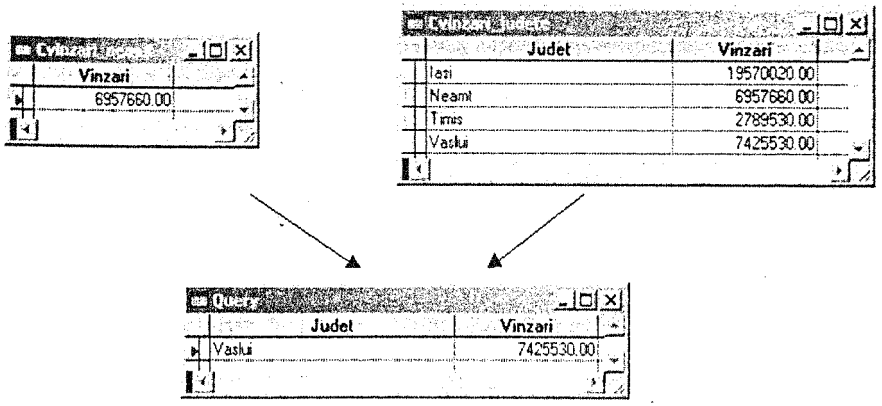


Figura 5.15. Determinarea județului cu vânzări imediat peste cele ale județului Neamț

Exemplul 22

Care este clientul cel mai mare datornic?

Listingul 5.6 conține programul ce răspunde la această problemă.

Listing 5.6. Clientul cu cel mai mare rest de plată

```
* Valorile facturate, pe clienti
SELECT CodCl, SUM(Cantitate * PretUnit * (1+ProcTVA)) AS Facturat ;
FROM FACTURI F ;
  INNER JOIN LINIIFACT LF ON F.NrFact=LF.NrFact ;
  INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
INTO CURSOR cFACTURAT          ;
GROUP BY CodCl

* Valorile incasate, pe clienti
SELECT CodCl, SUM(Transa) AS Incasat ;
FROM FACTURI F ;
  INNER JOIN INCASFACT I ON F.NrFact=I.NrFact ;
INTO CURSOR cINCASAT ;
GROUP BY CodCl

* Restul de plata, pe clienti
SELECT cFACTURAT.CodCl, Facturat, NVL(Incasat,0) AS Incasat, ;
  Facturat - NVL(Incasat,0) AS De_Incasat ;
FROM cFACTURAT LEFT OUTER JOIN cINCASAT ;
  ON cFACTURAT.CodCl=cINCASAT.CodCl ;
INTO CURSOR cDe_INCASAT        ;

* Finalul apoteotic
SELECT DenCl, cDe_INCASAT.* ;
FROM cDe_INCASAT INNER JOIN CLIENTI ;
  ON cDe_INCASAT.CodCl=CLIENTI.CodCl ;
WHERE De_Incasat >= ALL ;
                 (SELECT De_Incasat ;
                 FROM cDe_INCASAT)
```

5.6.2. Utilizarea cursoarelor pentru actualizări

În paragraful anterior au fost introduse patru câmpuri calculate, dar nici unul nu a fost adus „la zi", din cauza limitărilor comenzii `UPDATE` în VFP. Modalitățile în care se poate redacta programul de actualizare a fiecăruia dintre atribute sunt destul de eterogene, de la soluții „curat xBase-iste" la soluții „preponderent SQL-istice" (scuzați terminologia !). Iată în listingul 5.7 o variantă simplă în care valoarea cu TVA a fiecărei linii se caculează și se salvează în cursorul `cVal`, cursor care se parcurge secvențial, pentru fiecare linie a sa operându-se modificarea în înregistrarea din LINIIFACT corespondentă.

Listing 5.7. Prima variantă de actualizare a atributului `ValCuTVA`

```
*
SELECT NrFact, LF.CodPr, Cantitate * PretUnit * (1+ProcTVA) AS ValCuTVA ;
INTO CURSOR cVAL ;
FROM LINIIFACT LF INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr

SELECT cVal
SCAN
  UPDATE LINIIFACT ;
  SET ValCuTVA = cVal.ValcuTVA ;
  WHERE NrFact = cVal.NrFact AND CodPr = cVal.CodPr
ENDSCAN
```


Pentru comparație, în listingul 5.8 este prezentată o soluție ce combină folosirea cursorului cu indexul primar al tabelei LINIIFACT:

Listing 5.8. A doua variantă de actualizare a atributului `ValCuTVA`

```
*
SELECT STR(NrFact,8)+STR(Linie,2) AS Cheie, Cantitate * PretUnit * (1+ProcTVA) AS ValCuTVA ;
INTO CURSOR cVAL ;
FROM LINIIFACT LF INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr

SELECT cVal
SCAN
  SELECT LINIIFACT
  SEEK cVal.Cheie ORDER TAG primaru
  REPLACE ValCuTVA WITH cVal.ValCuTVA
  SELECT cVal
ENDSCAN
```

iar în listingul 5.9, o variantă strict xBase:

Listing 5.9. A treia variantă de actualizare a atributului `ValCuTVA`

```
*
IF !USED('LINIIFACT')
  USE LINIIFACT in 0
ENDIF

IF !USED('PRODUSE')
  USE PRODUSE in 0
ENDIF

SELECT LINIIFACT
SCAN
  SELECT PRODUSE
  SEEK LINIIFACT.CodPr ORDER TAG CodPr
  SELECT LINIIFACT
  REPLACE ValCuTVA WITH Cantitate * PretUnit * (1+PRODUSE.ProcTVA)
ENDSCAN
```

Programul din listingul 5.10 este cel care realizează calculul celor trei atribute, `LINIIFACT.ValCuTVA`, `FACTURI.ValTotala` și `FACTURI.ValIncasată`. Pentru a mai crește factorul interes, valoarea încasată se actualizează prin intermediul unui tablou (masiv) – `vIncasat`.

Listing 5.10. Totalul celor trei noi atribute

```
UPDATE facturi SET ValTotala = 0, ValIncasata = 0, Penalizari = 0, Reduceri = 0

UPDATE liniifact SET ValCuTva = 0

IF !USED('PRODUSE')
  USE PRODUSE IN 0
ENDIF

SELECT LINIIFACT
SCAN
  * se preia procentul TVA
  SELECT PRODUSE
  SEEK LINIIFACT.CodPr ORDER TAG CodPr

  * calculul si inlocuirea valorii cu TVA
  SELECT LINIIFACT
  REPLACE ValCuTVA WITH Cantitate * PretUnit * (1+PRODUSE.ProcTVA)

  * adaugarea valorii in tabela facturi
  SELECT FACTURI
  SEEK LINIIFACT.NrFact ORDER TAG NrFact
  REPLACE ValTotala WITH ValTotala + LINIIFACT.ValCuTVA

  SELECT LINIIFACT
ENDSCAN

* calculul valorii incasate pentru fiecare factura si salvarea intr-un masiv
SELECT FACTURI.NrFact, SUM(NVL(Transa,0)) AS ValIncasata ;
INTO ARRAY vIncasat ;
FROM FACTURI LEFT OUTER JOIN INCASFACT ON FACTURI.NrFact=INCASFACT.NrFact ;
GROUP BY FACTURI.NrFact

* actualizarea valorii incasate in tabela FACTURI
SELECT FACTURI
SCAN
  pozitie = ASCAN (vIncasat, FACTURI.NrFact)
  REPLACE ValIncasata WITH vIncasat (pozitie + 1)
ENDSCAN
```

5.6.3. Alte considerații privind cursoarele

Un cursor poate fi creat şi prin comanda `CREATE CURSOR`, caz în care tabela temporară este actualizabilă. Formatul comenzii prezintă clauze similare `CREATE TABLE`:

```
CREATE CURSOR alias_name
   (fname1 type [(precision [, scale])
[NULL | NOT NULL]
[CHECK lExpression [ERROR cMessageText]]
[DEFAULT eExpression]
[UNIQUE]
[NOCPTRANS]]
[, fname2 ...])
| FROM ARRAY ArrayName
```


O dată creat şi declarate restricțiile, un cursor poate fi actualizat ca orice tabelă obişnuită, prin comenzile SQL: `INSERT`, `UPDATE` şi `DELETE`, precum şi orice altă comandă de editare specifică VFP. De asemenea, orice încălcare a vreunei restricții definite va fi semnalizată. După cum vom vedea în capitolul 9, şi cursoarelor, similar tabelelor virtuale, li se pot vizualiza şi configura proprietățile prin funcțiile `CURSORGETPROP()` şi `CURSORSETPROP()`.

Conținutul unui cursor poate fi adăugat la înregistrările curente ale unei tabele prin comanda `APPEND FROM DBF('<NumeCursor>')`. Astfel, putem redacta o variantă mai simplă dar mai primitivă de calcul al valorilor celor trei atribute. Prin fraze `SELECT` construim cursoare cu aceleaşi structuri ca ale tabelelor FACTURI şi LINIIFACT, însă cu atributele `ValCuTVA`, `ValTotala` şi `ValIncasata` calculate. Apoi ştergem toate liniile celor două tabele şi adăugăm conținutul cursoarelor. Programul este cel din listingul 5.11.

Listing 5.11. O altă variantă pentru calculul valorilor celor trei noi atribute

```
* folosirea  APPEND FROM DBF(<cursor>)

* se construieste un cursor cu structura identica LINIIFACT
* si cu toate atributele completate/calculate corect
SELECT NrFact, Linie, LF.Codpr, Cantitate, ;
   PretUnit, Cantitate * PretUnit * (1+PrpcTVA) AS ValCuTVA ;
INTO CURSOR cLINIIFACT ;
FROM LINIIFACT LF INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr

* se goleste tabela LINIIFACT si apoi se preia continutul cursorului
DELETE FROM LINIIFACT
SELECT LINIIFACT
PACK
APPEND FROM DBF('cLINIIFACT')

* analog pentru tabela FACTURI

SELECT FACTURI.NrFact, DataFact, Gestiune, CodCl, Obs, ;
  SUM(NVL(ValCuTVA,0)) AS ValTotala, ;
  Reduceri, Penalizari, ;
   SUM(NVL(Transa,0)) AS ValIncasata ;
INTO CURSOR cFacturi ;
FROM FACTURI ;
  LEFT OUTER JOIN LINIIFACT ON FACTURI.NrFact = LINIIFACT.NrFact ;
  LEFT OUTER JOIN INCASFACT ON FACTURI.NrFact=INCASFACT.NrFact ;
GROUP BY FACTURI.NrFact

DELETE FROM FACTURI
SELECT FACTURI
PACK
APPEND FROM DBF('cFACTURI')
```

Cursoarele pot fi, în unele situații, un bun substitut al vectorilor. Să luăm un asemenea caz, cel al calculării reducerilor pentru plata rapidă a facturilor. Să presupunem că, pentru clienții buni platnici, se acordă reduceri la plată:

- pentru tranşele de facturi plătite (`INCASARI.DataInc`) în termen de până la 10 zile, acordăm o reducere de 10%;

- între 11 şi 12 zile procentul este de 9%;
- între 13 şi 15 zile – 8%;
- peste 16 zile – nu se acordă reduceri (eventual se calculează penalizări, dar asta-i altă poveste).

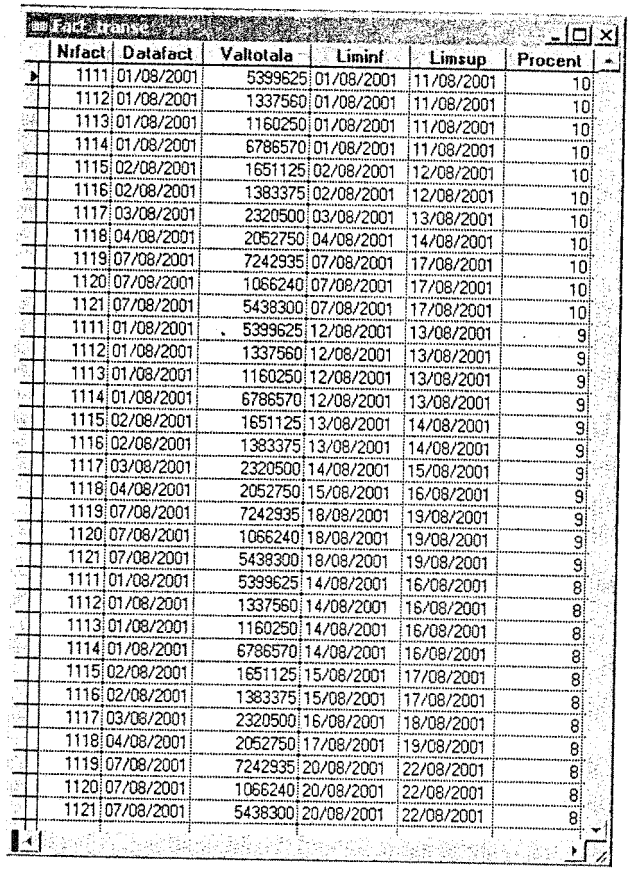
Tranşele de acordare a reducerilor pot fi stocate în cursoare şi, astfel, folosite în interogări. În listingul 5.12 este prezentat programul care realizează calculul reducerilor şi actualizarea atributului `FACTURI.Reduceri`.

Listing 5.12. Calculul reducerilor acordate pentru încasări

```
* se creaza si populeaza cursorul cu transele de reduceri
CREATE CURSOR transe_red ( ;
   zile_min INTEGER,  ;
   zile_max INTEGER,  ;
   procent INTEGER ;
   )

INSERT INTO transe_red VALUES ( 0,   10, 10)
INSERT INTO transe_red VALUES (11,   12,  9)
INSERT INTO transe_red VALUES (13,   15,  8)
INSERT INTO transe_red VALUES (16, 99999,  0)


* pentru fiecare factura cu valoare peste zero se insereaza
* in cursrul FACT_TRANSE cite o linie pentru fiecare transa
* posibila de reduceri
SELECT NrFact, DataFact, ValTotala, DataFact + zile_min AS LimInf, ;
   DataFact + zile_max AS LimSup, procent ;
FROM FACTURI, TRANSE_RED      ;
INTO CURSOR FACT_TRANSE ;
WHERE procent > 0 AND ValTotala > 0

* se calculeaza reducerile pe fiecare transa
SELECT FACT_TRANSE.*, DataInc, Transa, INT(Procent * Transa / 100) AS Reducere ;
INTO CURSOR reduceri_transe ;
FROM INCASFACT INNER JOIN INCASARI ON INCASFACT.CodInc=INCASARI.CodInc ;
   INNER JOIN FACT_TRANSE  ;
      ON INCASFACT.NrFact = FACT_TRANSE.NrFact  ;
WHERE INCASARI.DataInc >= FACT_TRANSE.LimInf AND ;
INCASARI.DataInc <= FACT_TRANSE.LimSup


* se actualizeaza atributul FACTURI.Reduceri
DIME suma_ (1,1)
SELECT facturi
SCAN
   suma_ = 0
   SELECT SUM(Reducere) ;
      INTO ARRAY suma_ ;
      FROM REDUCERI_TRANSE ;
      WHERE NrFact = facturi.Nrfact
   SELECT facturi
   REPLACE Reduceri WITH suma_(1,1)
ENDSCAN
```


Soluţia e destul de scumpă, dar operaţională. Tranşele de reduceri constituie obiectul cursorului actualizabil `TRANSE_RED`, actualizabil automat deoarece a fost creat prin comanda `CREATE CURSOR` şi nu printr-un `SELECT`.

Prima frază `SELECT` constituie unul dintre puţinele exemple de folosire a produsului cartezian. Cursorul (Read-Only) `FACT_TRANSE` conţine, pentru fiecare factură cu valoare mai mare decât zero, toate intervalele calendaristice pentru care se pot acorda reduceri. Ţinând cont de liniile tabelelor din baza de date, iată în figura 5.16 conţinutul cursorului.



Nrfact	Datafact	Valtotala	Liminf	Limsup	Procent
1111	01/08/2001	5399625	01/08/2001	11/08/2001	10
1112	01/08/2001	1337560	01/08/2001	11/08/2001	10
1113	01/08/2001	1160250	01/08/2001	11/08/2001	10
1114	01/08/2001	6786570	01/08/2001	11/08/2001	10
1115	02/08/2001	1651125	02/08/2001	12/08/2001	10
1116	02/08/2001	1383375	02/08/2001	12/08/2001	10
1117	03/08/2001	2320500	03/08/2001	13/08/2001	10
1118	04/08/2001	2052750	04/08/2001	14/08/2001	10
1119	07/08/2001	7242935	07/08/2001	17/08/2001	10
1120	07/08/2001	1066240	07/08/2001	17/08/2001	10
1121	07/08/2001	5438300	07/08/2001	17/08/2001	10
1111	01/08/2001	5399625	12/08/2001	13/08/2001	9
1112	01/08/2001	1337560	12/08/2001	13/08/2001	9
1113	01/08/2001	1160250	12/08/2001	13/08/2001	9
1114	01/08/2001	6786570	12/08/2001	13/08/2001	9
1115	02/08/2001	1651125	13/08/2001	14/08/2001	9
1116	02/08/2001	1383375	13/08/2001	14/08/2001	9
1117	03/08/2001	2320500	14/08/2001	15/08/2001	9
1118	04/08/2001	2052750	15/08/2001	16/08/2001	9
1119	07/08/2001	7242935	18/08/2001	19/08/2001	9
1120	07/08/2001	1066240	18/08/2001	19/08/2001	9
1121	07/08/2001	5438300	18/08/2001	19/08/2001	9
1111	01/08/2001	5399625	14/08/2001	16/08/2001	8
1112	01/08/2001	1337560	14/08/2001	16/08/2001	8
1113	01/08/2001	1160250	14/08/2001	16/08/2001	8
1114	01/08/2001	6786570	14/08/2001	16/08/2001	8
1115	02/08/2001	1651125	15/08/2001	17/08/2001	8
1116	02/08/2001	1383375	15/08/2001	17/08/2001	8
1117	03/08/2001	2320500	16/08/2001	18/08/2001	8
1118	04/08/2001	2052750	17/08/2001	19/08/2001	8
1119	07/08/2001	7242935	20/08/2001	22/08/2001	8
1120	07/08/2001	1066240	20/08/2001	22/08/2001	8
1121	07/08/2001	5438300	20/08/2001	22/08/2001	8

Figura 5.16. Cursorul `FACT_TRANSE`

A doua frază `SELECT` determină reducerile efective, pe tranşe, conţinutul cursorului creat – `REDUCERI_TRANSE` – fiind prezentat în figura 5.17.

Nrfact	Datafact	Valtotala	Liminf	Limsup	Procent	Datainc	Transa	Reducere
1120	07/08/2001	1066240	07/08/2001	17/08/2001	10	16/08/2001	731557	73155
1118	04/08/2001	2052750	15/08/2001	16/08/2001	9	16/08/2001	1026375	92373
1118	04/08/2001	2052750	15/08/2001	16/08/2001	9	15/08/2001	1026375	92373
1111	01/08/2001	5399625	14/08/2001	16/08/2001	8	15/08/2001	5399625	431970
1112	01/08/2001	1337560	14/08/2001	16/08/2001	8	15/08/2001	487705	39016
1117	03/08/2001	2320500	16/08/2001	18/08/2001	8	18/08/2001	369680	29574
1117	03/08/2001	2320500	16/08/2001	18/08/2001	8	17/08/2001	975410	78032
1117	03/08/2001	2320500	16/08/2001	18/08/2001	8	16/08/2001	975410	78032

Figura 5.17. Cursorul REDUCERI_TRANSE

În final, se însumează toate reducerile pe facturi şi se actualizează atributul FACTURI.Reduceri.

Varianta prezentată este una exemplificatoare (se observă că, pentru câteva facturi, valoarea încasată plus reducerile sunt peste valoarea facturii), dar poate fi adaptată la practica românească.

5.7. Variabile utilizate în consultări. Macrosubstituţie şi SQL

Utilizarea variabilelor în interogările SQL este utilă mai ales atunci când rapoartele sau informaţiile trebuie filtrate calendaristic, geografic etc.

5.7.1. Variabile pentru filtrarea interogărilor

Să începem cu un exemplu simplu. Într-o aplicaţie avem de listat facturile emise către un client pentru un anumit interval de timp definit printr-o dată iniţială şi o dată finală. Pentru aceasta se poate redacta un program 15_13.prg, care preia cei trei parametri (codul clientului, datele iniţială şi finală ale intervalului calendaristic) – listing 5.13.

Listing 5.13. Programul L5_13.PRG

```
PARAMETER codcl_, datai_, dataf_

SELECT DenCl AS Client, Nrfact, DataFact, Gestiune, ValTotala AS Valoare ;
FROM CLIENTI C LEFT OUTER JOIN FACTURI F ON C.CodCl = F.CodCl ;
WHERE F.CodCl = codcl_ AND Datafact BETWEEN datai_ AND dataf_
```

Pentru a obţine lista facturilor emise clientului 1 (cod 1001) în perioada 2-7 august 2001, se apelează programul de mai sus prin comanda:

```
DO 15_13 WITH 1001, {^2001/08/02}, {^2001/08/07}
```

Cu varianta din listingul 5.14 se poate ameliora afişarea pe ecran.

Listing 5.14. O afişare un pic mai plăcută

```
PARAMETER codcl_, datai_, dataf_

SELECT DenCl AS Client, Nrfact, DataFact, Gestiune, ValTotala AS Valoare ;
INTO CURSOR c1 ;
FROM CLIENTI C LEFT OUTER JOIN FACTURI F ON C.CodCl = F.CodCl ;
WHERE F.CodCl = codcl_ AND Datafact BETWEEN datai_ AND dataf_

titlu_ = 'Lista facturilor pentru clientul '+ALLTRIM(c1.Client)+' in perioada '+;
  DTOC(datai_)+' - '+DTOC(dataf_)

SELECT c1
BROWSE TITLE titlu_
USE
```

La execuţia comenzii

```
DO 15_14 WITH 1001, {^2001/08/02}, {^2001/08/07}
```

se va afişa o fereastră BROWSE de forma celei din figura 5.18.

Lista facturilor pentru clientul Client 1 SRL in perioada 02/08/2001 - 07/08/2001

Client	Nrfact	Datafact	Gestiune	Valoare
Client 1 SRL	1115	02/08/2001	002	1651125
Client 1 SRL	1117	03/08/2001	002	6961500
Client 1 SRL	1118	04/08/2001	003	4105500
Client 1 SRL	1120	07/08/2001	003	1066240

Figura 5.18. Interogare parametrizată şi titlu dinamic pentru fereastra BROWSE

Se mai cuvine de adăugat că, în afara filtrării şi denumirii „dinamice" a ferestrei de vizualizare a rezultatelor interogării, ultimul program „lasă curat" în urma sa, închizând cursorul la ieşirea din BROWSE.

5.7.2. Macrosubstituţia şi fraze SELECT

Să presupunem că o aceeaşi listă – a facturilor ce conţine următoarele date:

- numărul facturii,
- data emiterii,
- numele clientului,
- localitatea în care-şi are sediul clientul,
- valoarea încasată până în momentul curent –

trebuie obţinută pentru un anumit interval calendaristic, în orice ordine declarată prin toate combinaţiile posibile ale celor cinci câmpuri. Listingul 5.15 conţine o primă variantă de program.

Listing 5.15. Prima varinată de ordonare dinamică

```
PARAMETER datai_, dataf_, cimp1_, cimp2_, cimp3_, cimp4_

SELECT DenCl AS Client, Loc AS Localitate, Nrfact, DataFact, Gestiune, ValTotala AS Valoare ;
FROM CLIENTI C LEFT OUTER JOIN FACTURI F ON C.CodCl = F.CodCl ;
```

```
   INNER JOIN LOCALITATI L ON C.CodPost = L.CodPost ;
WHERE Datafact BETWEEN datai_ AND dataf_ ;
ORDER BY &cimp1_, &cimp2_, &cimp3_, &cimp4_
```

Clauza ORDER BY se construieşte dinamic cu ajutorul macrosubstituţiei. Invocarea acestui program se face cu o comandă DO de genul:

```
DO l5_15 WITH {^2001/08/02}, {^2001/08/07}, 'Loc',
 'DataFact', 'DenCl', 'ValTotala'
```

În continuare vă propunem o variantă mai elegantă. Ce-ar fi ca, în afara celor două date calendaristice, de început şi de sfârşit, ordonarea să fie precizată printr-o listă care să conţină unul, două, trei... sau chiar nici un atribut? Programul din listingul 5.16 preia un parametru de tip şir de caractere care este, de fapt, lista atributelor de ordonare, atribute separate prin virgulă. Aşa încât programul analizează virgulele din şir şi extrage atributele pe baza cărora se va construi, dinamic, clauza WHERE.

Listing 5.16. Se preia numai data iniţială şi finală, plus ordinea de prezentare, sub formă de listă de atribute separate prin virgulă

```
PARAMETER datai_, dataf_, ordinea_
i = 1
* se determina prima pozitie a virgulei (asta inseamna cu sunt macar
* doua atribute de ordonare)
poz_ = AT (',', ordinea_, i)

IF poz_ > 0                    && exista macar o virgula, deci sunt mai multe atribute de ordonare

  * se localizeaza toate virgulele si se extrag atributele cuprinse intre virgule
  poz_preced = 0
  DO WHILE poz_ # 0
     ii = str(i,1)
     cimp&ii = SUBSTR(ordinea_, poz_preced + 1, poz_ - poz_preced - 1)
     poz_preced = poz_
     i = i + 1
     poz_ = AT (',', ordinea_, i)
  ENDDO

  * a mai ramas atributul dintre ultima virgula si sfirsitul sirului de ordonare
  ii = str(i,1)
  cimp&ii = SUBSTR(ordinea_, poz_preced + 1, LEN(ordinea_) - poz_preced)

  * se construieste clauza ORDER BY
  order_by_ = ''
  FOR j = 1 TO i
     jj = STR(j,1)
     IF j > 1
        order_by_ = order_by_ + ', '
     ENDIF
     order_by_ = order_by_ + ALLTRIM(cimp&jj)
  ENDFOR

ELSE  && exista maximum un atribut de ordonare
```



```
IF LEN(ALLT(ordinea_)) = 0                    && de fapt, nu exista nici un atribut de ordonare
     order_by_ = ' 1 ' && ordonarea se face dupa prima coloana
  ELSE
     order_by_ = ALLTRIM(ordinea_)&& exista un singur atribut de ordonare
  ENDIF

ENDIF

* si acum, interogarea !
SELECT DenCl AS Client, Loc AS Localitate, Nrfact, DataFact, Gestiune, ValTotala AS Valoare ;
FROM CLIENTI C LEFT OUTER JOIN FACTURI F ON C.CodCl = F.CodCl ;
  INNER JOIN LOCALITATI L ON C.CodPost = L.CodPost ;
WHERE Datafact BETWEEN datai_ AND dataf_ ;
ORDER BY &order_by_
```

Iată şi câteva moduri de apel a programului:

```
DO l5_16 WITH {^2000/08/02}, {^2001/08/20}, 'valtotala,
     dencl, nrfact'

DO l5_16 WITH {^2000/08/02}, {^2001/08/20}, 'nrfact'

DO l5_16 WITH {^2000/08/02}, {^2001/08/20}, 'valtotala,
     nrfact'

DO l5_16 WITH {^2000/08/02}, {^2001/08/20}, ''
```

5.7.3. Calculul automat al valorilor unor atribute agregate

Practica dezvoltării aplicaţiilor de lucru cu baze de date a consacrat sintagma *denormalizare*. Frecvent, în tabele sunt incluse atribute redundante, calculate prin însumarea valorilor unor atribute din tabelele-copil. Forţând un pic lucrurile, ne propunem să construim o rutină generalizată căreia să-i trimitem ca parametri numele tabelelor-părinte şi copil, numele atributului din părinte ce trebuie actualizat, numele atributului din copil pe baza căruia se face actualizarea, precum şi atributul-cheie al tabelei-părinte.

Listing 5.17. Generalizarea actualizării valorilor unui câmp calculat

```
*
PARAMETER parinte_, cheie_parinte_, cimp_parinte_, copil_, cheie_copil_, cimp_copil_

* se zerorizeaza cimpul ce urmeaza a fi calculat
UPDATE (parinte_) SET &cimp_parinte_ = 0

* se declara masivul in care va fi preluata valoarea calculata
DIME suma_ (1,1)

SELECT (parinte_)
SCAN
  suma_ = 0
  * valoarea cheii pimare a liniei curente din tabela parinte
  val_cheie_parinte_ = &cheie_parinte_
```

```
   * insumarea liniilor-copil
   SELECT SUM(&cimp_copil_) INTO ARRAY suma_ FROM (copil_) ;
 WHERE &cheie_copil_ = val_cheie_parinte_

   * inlocuirea valorii calculate in linia curenta a tabelei parinte
   SELECT (parinte_)
   REPLACE &cimp_parinte_ WITH suma_(1,1)
 ENDSCAN
```

Pentru calcularea valorilor atributului FACTURI.ValTotală pe baza liniilor-copil din LINIIFACT, atributul-copil fiind LINIIFACT.ValCuTVA, se invocă programul de mai sus sub forma:

```
do l5_17 with 'facturi', 'nrfact', 'valtotala', 'liniifact',
   'nrfact', 'valcutva'
```

Însumarea tranşelor de încasare (INCASFACT.Transa) pentru fiecare factură şi, astfel, calcularea valorilor atributului FACTURI.ValIncasata se va face astfel:

```
do l5_17 with 'facturi', 'nrfact', 'valincasata',
 'incasfact', 'nrfact', 'transa'
```

5.7.4. Macrosubstituţie şi iteraţii SQL

Tot cu ajutorul macrosubstituţiei se pot rezolva situaţii dintre cele mai dificile de obţinere a unor rapoarte pretenţioase. Spre exemplificare, dorim obţinerea unui raport matriceal în care, pe verticală, să fie prezente toate produsele comercializate de firmă, iar „ordonata" să fie constituită din primele patru zile ale lunii august 2001 – ca în figura 5.19.

Vinzari - Produse/Zile

	Denpr	Codpr	Zi_01_08_2001	Zi_02_08_2001	Zi_03_08_2001	Zi_04_08_2001
▶	Produs 1	1	595000	0	1130500	1660050
	Produs 2	2	3969245	3034500	1190000	392700
	Produs 3	3	357000	0	0	0
	Produs 4	4	564060	0	0	0
	Produs 5	5	9198700	0	0	0

Figura 5.19. O situaţie matriceală

Pentru facilitarea prezentării, ne-am limitat la patru zile, dar programul din listingul 5.18 funcţionează pentru oricâte zile, singura problemă fiind popularea cursorului ZILE. În programul nostru, aceasta se face prin INSERT-uri directe, însă în aplicaţii datele iniţiale şi finale ale intervalului se pot prelua printr-un formular şi, astfel, popularea cursorului se automatizează.

Listing 5.18. Situaţie matriceală a vânzării produselor pe primele patru zile ale lunii august 2001

```
SET MARK TO '_'                    && separatorul ptr. data calendaristica

CREATE CURSOR Zile (Zi DATE)
INSERT INTO Zile VALUES ({^2001/08/01})
INSERT INTO Zile VALUES ({^2001/08/02})
```

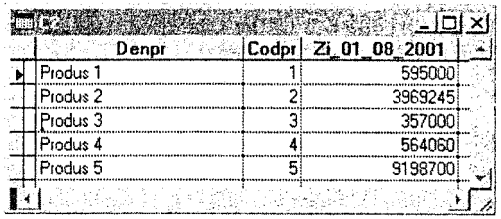


```
INSERT INTO Zile VALUES ({^2001/08/03})
INSERT INTO Zile VALUES ({^2001/08/04})

SELECT DataFact, DenPr, LF.CodPr, SUM(INT(Cantitate * PretUnit * (1+ProcTVA))) AS Valoare ;
INTO CURSOR Zile_Produse ;
FROM FACTURI F INNER JOIN LINIIFACT LF ON F.NrFact = LF.NrFact ;
  INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
WHERE DataFact IN ;
  (SELECT Zi ;
   FROM Zile) ;
GROUP BY DataFact, DenPr, LF.CodPr

SELECT DenPr, CodPr ;
INTO CURSOR c1 ;
FROM PRODUSE ;
ORDER BY DenPr

i = 1
SELECT Zile
SCAN

  ii = ALLT(STR(i,2))
  ii_plus_1 = ALLT(STR(i+1,2))
  ziua_ = '_'+DTOC(Zile.Zi)

  SELECT c&ii..*, NVL(ZILE_PRODUSE.Valoare,0) AS Zi&ziua_ ;
  FROM c&ii LEFT OUTER JOIN ZILE_PRODUSE ;
     ON c&ii..CodPr = ZILE_PRODUSE.CodPr AND DataFact = Zile.Zi ;
  INTO CURSOR c&ii_plus_1
  i = i + 1
  SELECT Zile

ENDSCAN

SELECT c&ii_plus_1
BROWSE TITLE 'Vinzari - Produse/Zile'

SELECT Zile
USE
SELECT Zile_Produse
USE

FOR j = 1 TO i
  ii = ALLT(STR(j,2))
  SELECT c&ii
  USE
ENDFOR
```

Câteva explicaţii n-ar strica în acest moment. Cursorul ZILE conţine toate datele calendaristice care trebuie să apară drept coloane ale raportului. Valorile vânzărilor pe produse şi zile sunt calculate şi stocate în cursorul ZILE_PRODUSE. Liniile raportului sunt alcătuite din toate produsele firmei, astfel încât cursorul „de start" – c1 – este obţinut din tabela PRODUSE.

Numărul iteraţiilor depinde de numărul înregistrărilor din cursorul ZILE – numărul zilelor pentru care intersează raportul. Corpul buclei conţine joncţiunea cursorului „curent" cu ziua corespunzătoare înregistrării curente din cursorul ZILE. Astfel, dacă c1 nu are

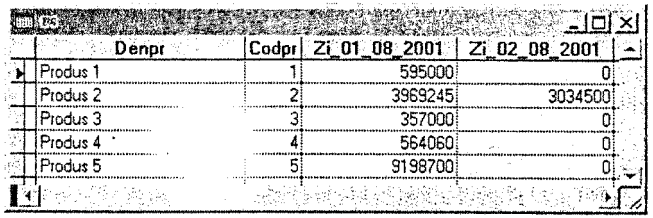
nevoie de explicaţii, la prima parcurgere a corpului buclei se obţine cursorul c2, care, pe lângă cele două coloane din c1, conţine un atribut corespunzător primei date din ZILE (1 august 2001) – vezi figura 5.20.



Denpr	Codpr	Zi_01_08_2001
Produs 1	1	595000
Produs 2	2	3969245
Produs 3	3	357000
Produs 4	4	564060
Produs 5	5	9198700

Figura 5.20. Cursorul c2

A doua trecere prin buclă se soldează cu un nou cursor – c3 –, vizualizat în figura 5.21.



Denpr	Codpr	Zi_01_08_2001	Zi_02_08_2001
Produs 1	1	595000	0
Produs 2	2	3969245	3034500
Produs 3	3	357000	0
Produs 4	4	564060	0
Produs 5	5	9198700	0

Figura 5.21. Cursorul c3

Ultimul cursor este identic celui din figura 5.19. După BROWSE-ul vizualizator, se face curăţenie în zonele de lucru, închizându-se cursoarele deschise şi care, la un număr mare de zile, ocupă un volum important de memorie.

Este drept că programul poate fi simplificat folosind facilitatea salvării rezultatelor consultării în acelaşi cursor din care se extrag datele – vezi listingul 5.19.

Listing 5.19. Versiunea 2 a situaţiei matriceale a vânzării produselor pe primele patru zile ale lunii august 2001

```
*
* ... identic cu l5.18
*

SELECT DenPr, CodPr ;
INTO CURSOR c1 ;
FROM PRODUSE ;
ORDER BY DenPr

SELECT Zile
SCAN
  ziua_ = '_'+DTOC(Zile.Zi)

  SELECT c1.*, NVL(ZILE_PRODUSE.Valoare,0) AS Zi&ziua_  :
  FROM c1 LEFT OUTER JOIN ZILE_PRODUSE ;
      ON c1.CodPr = ZILE_PRODUSE.CodPr AND DataFact = Zile.Zi ;
  INTO CURSOR c1
```



```
  SELECT Zile

ENDSCAN

SELECT c1
BROWSE LAST title 'Vinzari - Produse/Zile'

SELECT Zile
USE
SELECT Zile_Produse
USE
```

Capitolul 6

Proceduri stocate

Procedurile stocate reprezintă un modul esențial al oricărei aplicații profesionale ce utilizează baze de date. Altădată opțiune doar a serverelor de baze de date, procedurile stocate sunt prezente în majoritatea versiunilor actuale ale SGBD-urilor, de la Visual la FoxPro (încă din VFP 3 – 1994-1995) și Access (abia în versiunea 2000-1999) până la PostgreSQL. Toate procedurile stocate se regăsesc în dicționarul de date care, în VFP, este alcătuit din trei fișiere, cel mai important având extensia `.DBC`.

6.1. Dicționarul de date

Atât obiectele, cât mai ales dicționarul bazei prezintă diferențe semnificative de la SGBD la SGBD. Dacă e să începem cu una dintre cele mai simple forme de dicționar de date, putem face referire la containerul (*database container*) Visual FoxPro. Acesta este o tabelă specială cu structura prezentată în figura 6.1.

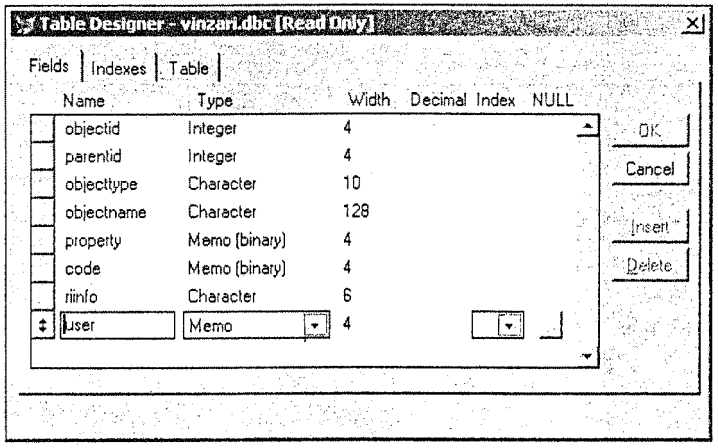


Figura 6.1. Structura containerului bazei de date în VFP

Fiecare obiect din bază este identificat printr-un număr întreg unic – `objectid`. Ierarhia obiectelor (de exemplu, atributele unei tabele) este reflectată prin câmpul `parentid`. Tipologia obiectelor din container (atributul `objecttype`) este: *Database*, *Table*, *View*, *Field*, *Index* și *Relation*. Dacă primele nu necesită comentarii, despre *Relation* trebuie spus că reflectă o legătură permanentă între două tabele, legătură ce nu asigură automat și restricțiile referențiale. Procedurile stocate nu sunt un tip special, ci au asociate

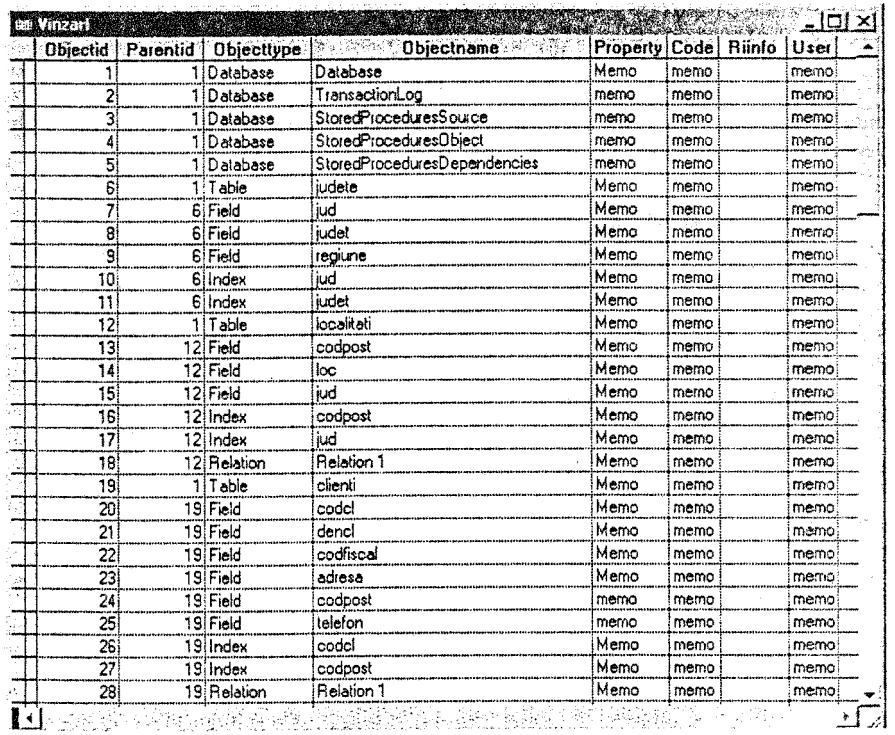
trei obiecte, `StoredProceduresSource`, `StoredProceduresObject` și `StoredProceduresDependencies`, toate de tipul *Database*.

Deschiderea prin comanda `USE` a containerului ridică ceva probleme (baza de date trebuie să fie închisă). În schimb, se poate folosi `Data Session` (fie din meniu – `Window→Data Session`, fie prin comanda `SET`), mai precis `Open→Other...→ Files of Type: All Files (*.*)→Name: vinzari.DBC`.

O dată deschisă, tabela-dicționar poate fi referită printr-o comandă `SELECT` (xBase):

```
SELECT vinzari
BROWSE
```

O parte din conținutul dicționarului bazei de date VINZARI este prezentată în figura 6.2.



Vinzari

Objectid	Parentid	Objecttype	Objectname	Property	Code	Riinfo	User
1	1	Database	Database	Memo	memo		memo
2	1	Database	TransactionLog	memo	memo		memo
3	1	Database	StoredProceduresSource	memo	memo		memo
4	1	Database	StoredProceduresObject	memo	memo		memo
5	1	Database	StoredProceduresDependencies	memo	memo		memo
6	1	Table	judete	Memo	memo		memo
7	6	Field	jud	Memo	memo		memo
8	6	Field	judet	Memo	memo		memo
9	6	Field	regiune	Memo	memo		memo
10	6	Index	jud	Memo	memo		memo
11	6	Index	judet	Memo	memo		memo
12	1	Table	localitati	Memo	memo		memo
13	12	Field	codpost	Memo	memo		memo
14	12	Field	loc	Memo	memo		memo
15	12	Field	jud	Memo	memo		memo
16	12	Index	codpost	Memo	memo		memo
17	12	Index	jud	Memo	memo		memo
18	12	Relation	Relation 1	Memo	memo		memo
19	1	Table	clienti	Memo	memo		memo
20	19	Field	codcl	Memo	memo		memo
21	19	Field	dencl	Memo	memo		memo
22	19	Field	codfiscal	Memo	memo		memo
23	19	Field	adresa	Memo	memo		memo
24	19	Field	codpost	memo	memo		memo
25	19	Field	telefon	memo	memo		memo
26	19	Index	codcl	Memo	memo		memo
27	19	Index	codpost	Memo	memo		memo
28	19	Relation	Relation 1	Memo	memo		memo

Figura 6.2. Prima parte din tabela-dicționar `VINZARI.DBC`

Obiectul-rădăcină este însăși baza de date (`Objectid` – 1, `Objecttype` – `Database`, `Objectname` – `DataBase`), care este părinte pentru următoarele obiecte:

- jurnalul tranzacțiilor – `TransactionLog`;
- formatul sursă al procedurilor stocate – `StoredProceduresSource`;
- formatul compilat al procedurilor stocate – `StoredProceduresObject`;

- depedenţele procedurilor stocate – `StoredProceduresDependencies`;
- toate tabelele bazei (`Objecttype – Table`);
- toate tabelele derivate locale ale bazei (`Objecttype – View`);
- toate conexiunile la alte servere de BD (`Objecttype – Connection`);
- toate tabelele derivate „la distanţă" – create prin legarea de tabele ale BD gestionate prin alte SGBD-uri (`Objecttype – View`).

În general nu se recomandă deschiderea tabelei-dicţionar, ci obţinerea informaţiilor pe care le conţine prin consultări.

6.1.1. Informaţii despre tabele

Dacă se doreşte extragerea denumirii tuturor tabelelor din bază, se poate folosi o consultare relativ simplă:

```
SELECT * FROM vinzari.dbc WHERE OBJECTTYPE='Table'
```

Răspunsul este cel din figura 6.3. Atributul `Property` furnizează informaţii precum numele fişierului `.DBF` de pe disc asociat tabelei, indexul primar al tabelei şi numele eventualelor declanşatoare ale tabelei.

Query

Objectid	Parentid	Objecttype	Objectname	Property	Code	Riinfo	User
6	1	Table	judete	Memo	memo		memo
12	1	Table	localitati	Memo	memo		memo
19	1	Table	clienti	Memo	memo		memo
29	1	Table	persoane	Memo	memo		memo
43	1	Table	persclienti	Memo	memo		memo
52	1	Table	produse	Memo	memo		memo
60	1	Table	gestiuni	Memo	memo		memo
73	1	Table	facturi	Memo	memo		memo
79	1	Table	liniifact	Memo	memo		memo
85	1	Table	incasari	Memo	memo		memo
92	1	Table	incasfact	Memo	memo		memo

Figura 6.3. Extragerea din containerul BD a informaţiilor despre tabele

Pe baza consultării tabelei-container, se poate redacta un program plasat la deschiderea oricărei aplicaţii, program care deschide toate tabelele bazei şi declară primul eventual index al tabelei drept index curent (activ) – listing 6.1.

Listing 6.1. Deschiderea automată a tabelelor BD şi stabilirea indexului activ

```
* se presupune ca baza e deja deschisa
SELECT left(objectname,30) as tabela ;
FROM vinzari.dbc ;
INTO CURSOR cTabele ;
WHERE objecttype = "Table"

SELECT cTabele
SCAN
    tab_ = UPPER(allt(tabela))
    * daca tabela nu este deja deschisa, se asociaza unei zone de lucru
    IF !USED((tab_))
        USE (tab_) IN 0 EXCLUSIVE
    ENDIF

    * daca exista macar un index, primul (dintre indecsi) devine principal
    SELECT (tab_)
    IF TAGCOUNT() > 0
        SET ORDER TO TAG 1
    ENDIF
    SELECT cTabele
ENDSCAN
```


Lista tabelelor din bază poate fi obţinută şi sub forma unui masiv unidimensional (vector) prin funcţia `ADBOBJECTS(array, "TABLE")`. `Array` reprezintă numele masivului unde vor fi stocate denumirile tabelelor, iar a doua opţiune poate fi, în afară de `TABLE`, `VIEW`, `CONNECTION` şi `RELATION`, asupra cărora vom mai reveni. În aceste condiţii, programul se simplifică în oarecare măsură – vezi listingul 6.2.

Listing 6.2. Deschiderea tabelelor BD şi stabilirea indexului activ – varianta 2

```
* se presupune ca baza e deja deschisa
*
ADBOBJECTS(vTabele, "TABLE")
FOR i = 1 TO ALEN(vTabele)
    tab_ = vTabele(i)
    * daca tabela nu este deja deschisa, se asociaza unei zone de lucru
    IF !USED((tab_))
        USE (tab_) IN 0 EXCLUSIVE
    ENDIF

    * daca exista macar un index, primul (dintre indecsi) devine principal
    SELECT (tab_)
    IF TAGCOUNT() > 0
        SET ORDER TO TAG 1
    ENDIF
ENDFOR
```

Pentru a obţine numele indexului primar al unei tabele, se poate folosi atât atributul `Property` din înregistrarea corepunzătoare tabelei în dicţionarul de date, ceea ce reclamă o serie de artificii, dat fiind formatul de prezentare a valorilor acestui atribut, dar şi o serie de funcţii. Dacă presupunem că orice tabelă are index primar şi acesta este primul creat, funcţia `TAG(1)` întoarce tocmai numele acestuia.

Riscul e totuşi mare, deoarece pot exista tabele fără indecşi primari sau indexul primar poate fi creat cronologic după unul obişnuit (regular). Aşa încât este mult mai nimerit să se folosească o funcţie foarte puternică a VFP – `DBGETPROP()` –, ce furnizează o largă paletă de informaţii privitoare la obiectele bazei de date. Afişarea numelui indexului primar al tabelei PERSCLIENTI se realizează (şi) astfel:

```
? DBGETPROP('persclienti', 'TABLE','PrimaryKey')
```

Dacă interesează cheia de indexare, adică expresia ce reprezintă cheia primară a tabelei, atunci trebuie să apelăm la funcţia `KEY()`, care nu are drept argument numele indexului elementar (tag), ci numărul acestuia, număr determinat de funcţia `TAGNO()`. Astfel,

```
? TAGNO('primaru', 'PERSCLIENTI', 'PERSCLIENTI')
```

afişează 1 (al doilea 'PERSCLIENTI' asigură rezultatul corect, indiferent de zona de lucru curentă). Pentru generalizare, putem introduce funcţia DBGETPROP() ca argument al funcţie TAGNO():

```
? TAGNO(DBGETPROP('persclienti', 'table','PrimaryKey'),
'persclienti', 'persclienti')
```

În fine, expresia de indexare a tabelei PERSCLIENTI se obţine, indiferent de zona curentă de lucru, astfel:

```
? KEY(TAGNO(DBGETPROP('persclienti', 'table','PrimaryKey'),
'persclienti', 'persclienti'), 'persclienti')
```

6.1.2. Informaţii despre atribute

Şi informaţiile despre atributele tabelelor pot fi obţinute prin consultarea containerului bazei. Pentru a afişa (în clar) şi numele tabelelor din care fac parte, se foloseşte o joncţiune a tabelei-dicţionar cu ea însăşi:

```
SELECT v2.objectname AS Tabela, v1.objectid AS IdAtribut, ;
       v1.objectname AS Atribut, left(v1.property, 250) ;
       AS ProprietatiAtribut ;
FROM vinzari.dbc v1 INNER JOIN vinzari.dbc v2 ;
   ON v1.parentid=v2.objectid ;
WHERE v2.OBJECTTYPE='Table' AND v1.OBJECTTYPE='Field' ;
ORDER BY v1.objectid
```

Din păcate, modul de prezentare a valorilor implicite, regulilor de validare şi mesajelor de eroare afişate la încălcarea regulilor de validare la nivel de câmp este destul de criptic şi dificil de folosit în aplicaţii, după cum se observă în figura 6.4.

Tabela	Idatribut	Atribut	Proprietatiatribut
judete	7	jud	IIIIIII(jud=LTRIM(UPPER(jud)))|7IIIIII"Indicativul judetului se scrie cu majuscule !"|
judete	8	judet	$IIIIII(judet=LTRIM(PROPER(judet)))IIIIIII"Prima litera din fiecare cuvint al"+CHR(13)+"denumirii judetului este maj
judete	9	regiune	IIIIIII"Moldova"|2IIIIIII(INLIST(regiune,"Banat","Transilvania","Dobrogea","Oltenia","Muntenia","Moldova"))IIIIIII"
localitati	13	codpost	IIIIII(codpost=LTRIM(codpost))|8IIIIII"Codul postal se scrie fara spatii la inceput !"|
localitati	14	loc	IIIIII(loc=LTRIM(PROPER(loc)))IIIIIII"Prima litera din fiecare cuvint al"+CHR(13)+"denumirii localitatii este majusc
localitati	15	jud	IIIIIII"1S"|
clienti	20	codcl	IIIIIII(codcl>1000)|
clienti	21	dencl	4IIIIII(SUBSTR(dencl,1,1)=UPPER(SUBSTR(dencl,1,1)))|Y IIIIII"Prima litera din denumirea clientului este : "+CHR(
clienti	22	codfiscal	<IIIIIII(SUBSTR(codfiscal,1,1)=UPPER(SUBSTR(codfiscal,1,1)))|IIIIIII"Prima (eventuala) litera din codul fiscal : "+C
clienti	23	adresa	6IIIIII(SUBSTR(adresa,1,1)=UPPER(SUBSTR(adresa,1,1)))|W IIIIII"Prima (eventuala) litera din adresa : "+CHR(13
clienti	24	codpost	
clienti	25	telefon	
persoane	30	cnp	IIIIIII(cnp=LTRIM(UPPER(cnp)))|8IIIIII"Codul numeric personal se scrie fara spatii la inceput !"|
persoane	31	nume	"IIIIIII(nume=LTRIM(PROPER(nume)))|uIIIIII"Prima litera din fiecare cuvint al"+CHR(13)+"numelui este majuscula;"
persoane	32	prenume	(IIIIII(prenume=LTRIM(PROPER(prenume)))|xIIIIII"Prima litera din fiecare cuvint al"+CHR(13)+"prenumelui este ma
persoane	33	adresa	6IIIIII(SUBSTR(adresa,1,1)=UPPER(SUBSTR(adresa,1,1)))|>IIIIII"Prima litera din adresa este obligatoriu majuscul
persoane	34	sex	IIIIIII"B"|IIIIIII(INLIST(sex,"F","B"))|bIIIIII"Atributul Sex poate avea valorile F (de la Femeiesc)"+CHR(13)+"sau B (
persoane	35	codpost	
persoane	36	telacasa	

Figura 6.4. Modul de prezentare a informaţiilor despre atribute

Astfel încât, de multe ori, informaţiile legate de reguli de validare, valori implicite se obţin tot cu DBGETPROP(). Câteva exemple sunt grăitoare:

- afişarea valorii implicite a atributului JUDETE.Regiune:

```
? DBGETPROP('JUDETE.Regiune', 'Field', 'DefaultValue')
```

- aflarea regulii de validare pentru atributul JUDETE.Regiune:

```
? DBGETPROP('JUDETE.Regiune', 'Field', 'RuleExpression')
```

- afişarea mesajului de eroare la încălcarea regulii atributului JUDETE.Regiune:

```
? DBGETPROP('JUDETE.Regiune', 'Field', 'RuleText')
```

O altă soluţie simplă constă în folosirea comenzii COPY STRUCTURE EXTENDED. Astfel, comenzile:

```
SELECT judete
  COPY STRUCTURE EXTENDED TO tAtribute
```

au ca efect crearea unei tabele libere (neinclusă în baza de date) denumite tAtribute, care va conţine informaţii atât despre atribute, cât şi despre regulile de validare şi declanşatoarele tabelei, după cum se observă în figura 6.5.

Field_nam	Field_type	Field_len	Fiel	Field_null	Fi	Field_defa	Field_rule	Field_err	Table_rule	Table_err	Table_name	Ins_trig	Upd_trig
JUD	C	2	0	F	F	memo	Memo	Memo	memo	memo	judete	memo	memo
JUDET	C	25	0	F	F	memo	Memo	Memo	memo	memo		memo	memo
REGIUNE	C	15	0	F	F	Memo	Memo	Memo	memo	memo		memo	memo

Figura 6.5. O parte din informaţiile obţinute prin COPY STRUCTURE EXTENDED

O altă modalitate de a obţine informaţii despre atributele unei tabele, de data aceasta sub formă de masiv de memorie, ţine de folosirea funcţiei AFIELDS(). Astfel, prin:

```
AFIELDS(mAtribute, 'judete')
```

se creează masivul mAtribute, care va avea câte o linie pentru fiecare atribut al tabelei JUDETE şi 16 coloane în care se regăsesc aceleaşi informaţii ca în cazul comenzii COPY STRUCTURE EXTENDED:

- numele atributului;
- tipul;
- lungimea;
- numărul de poziţii fracţionare;
- dacă acceptă valori nule;
- dacă se acceptă modificarea codului de pagină;
- restricţia (regula de validare);
- mesajul de eroare afişat la încălcarea restricţiei;

- valoarea implicită;
- restricţia (regula de validare) la nivel de tabelă (înregistrare);
- mesajul de eroare afişat la încălcarea restricţiei la nivel de tabelă;
- numele lung al tabelei (altul decât cel al fişierului .DBF);
- declanşatorul de inserare;
- declanşatorul de modificare;
- declanşatorul de ştergere;
- comentarii despre tabele.

Rezultatul întors de funcţia AFIELDS() este numărul de atribute al tabelei.

6.1.3. Reflectarea în containerul bazei a relaţiilor permanente dintre tabele

Unul dintre tipurile de obiecte ale tabelei-dicţionar este Relation. În figura 6.2 analizăm obiectul cu identificatorul 18 (ObjectId - 18; ParentId - 12; ObjectType - Relation; ObjectName - Relation 1). În dreptul coloanei Property apare scris Memo, ceea ce semnifică, pe de o parte, că este un câmp de tip memo, iar pe de altă parte, primul M fiind majusculă, înseamnă că valoarea acestui câmp nu este vidă. Din păcate, la un dublu-click, valoarea câmpului Property este afişată într-un format nu tocmai dintre cele mai limpezi cu putinţă – vezi figura 6.6.

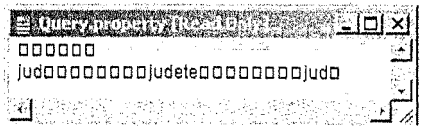


Figura 6.6. Informaţii „tulburi" despre relaţia permanentă dintre tabelele LOCALITATI şi JUDETE

Tabela curentă este LOCALITATI; prin urmare, cu puţină intuiţie, putem realiza că primul JUD reprezintă indexul acestei tabele prin care se stabileşte relaţia, JUDETE este numele tabelei-părinte, iar al doilea JUD – numele indexului tabelei-părinte prin care se realizează legătura permanentă.

Obţinerea acestor elemente în variabile sau tabele/cursoare presupune folosirea unor proceduri/funcţii de „des-catenare", aşa că vom căuta modalităţi mai simple. Revenim, după cum am promis, la funcţia ADBOBJECTS().

Funcţia ADBOBJECTS(mRelatii, "RELATION") conduce la obţinerea unui masiv de memorie denumit mRelatii. Masivul va avea atâtea linii câte relaţii permanente au fost declarate în baza de date şi cinci coloane, dintre care primele patru sunt cele care ne interesează în mod deosebit:

- numele tabelei-copil;
- indexul-copil implicat în relaţie;
- tabela-părinte;
- numele indexului din tabela-părinte (de obicei, indexul primar).

A cincea coloană nu este mai puţin importantă şi arată regulile instituite pentru păstrarea integrităţii referenţiale între tabelele implicate în relaţie:

- primul caracter se referă la regula aplicată la modificarea unei înregistrări-părinte, fiind I pentru Ignore, C pentru Cascade şi R pentru Restrict;
- al doilea caracter se referă la regula aplicabilă la ştergerea unei înregistrări-părinte (poate avea aceleaşi trei valori I/C/R);
- al treilea caracter priveşte regula aplicabilă la inserarea sau modificarea unei înregistrări-copil (poate avea numai două valori I/R).

Conţinutul masivului mRelatii poate fi „pasat" uşor unui cursor, după cum se observă în listingul 6.4 (conţinutul cursorului este prezentat în figura 6.7).

Listing 6.3. Informaţii despre relaţiile permanente stocate într-un cursor

```
CREATE CURSOR cRelatii (;
    tabela_copil CHAR(20), ;
    tabela_parinte CHAR(20), ;
    tag_copil CHAR(20), ;
    tag_parinte CHAR(20), ;
    reguli_udi CHAR(5) ;
    ) ;

ADBOBJECTS(mRelatii, 'Relation')

SELECT cRelatii
APPEND FROM ARRAY mRelatii
```

Crelatii

Tabela_copil	Tabela_parinte	Tag_copil	Tag_parinte	Reguli_udi
PERSCLIENTI	PERSOANE	CNP	CNP	CRR
PERSCLIENTI	CLIENTI	CODCL	CODCL	CRR
FACTURI	CLIENTI	CODCL	CODCL	CRR
LINIIFACT	PRODUSE	CODPR	CODPR	CRR
INCASFACT	INCASARI	CODINC	CODINC	CRR
LOCALITATI	JUDETE	JUD	JUD	CRR
CLIENTI	LOCALITATI	CODPOST	CODPOST	CRR
PERSOANE	LOCALITATI	CODPOST	CODPOST	CRR
GESTIUNI	LOCALITATI	CODPOST	CODPOST	CRR
GESTIUNI	PERSOANE	CNP	CNP	CRR
FACTURI	GESTIUNI	GESTIUNE	GESTIUNE	CRR
INCASFACT	FACTURI	NRFACT	NRFACT	CRR
LINIIFACT	FACTURI	NRFACT	NRFACT	CRR

Figura 6.7. Cursor obţinut dintr-un masiv creat prin funcţia ADBOBJECTS()

6.2. Tipologia şi aria de utilizare a procedurilor stocate

O procedură stocată este orice modul de cod, procedură sau funcţie depozitat în containerul (dicţionarul) bazei de date. Prin urmare, orice procedură stocată face parte din schema bazei şi, după cum vom vedea, se va altera sau distruge o dată cu alterarea/ştergerea dicţionarului.

Scrierea unei noi proceduri/funcţii stocate sau modificarea alteia existente se poate face astfel:

- din proiectantul bazei (`Database Designer`) se alege, fie din meniul general `Database`, fie din meniul contextual, opţiunea `Edit Stored Procedures...`
- din proiect se vor expanda nodurile `Data→Database→vinzari→Stored Procedures` şi apoi butonul de comandă `Modify`.
- prin comanda `MODIFY PROCEDURE`.

Fie că este vorba de o procedură, fie de o funcţie (VFP este destul de larg la inimă, nefăcând mare caz pe diferenţa dintre procedură şi funcţie), începutul acesteia este marcat prin cuvântul-cheie `PROCEDURE nume`, iar sfârşitul este marcat fie prin `ENDPROC`, fie prin `RETURN`.

Aria de utilizare a procedurilor stocate este una suficient de largă. Deşi tentaţia este mare, nu trebuie exagerat nici cu numărul lor, nici cu întinderea. Spunem asta ţinând seama că VFP nu are un mecanism coerent de recuperare a datelor în caz de accidente (până de curent, blocări ale sistemului de operare – chiar dacă în NT/2000 problemele de acest gen sunt cu mult mai rare).

Pe parcursul acestui capitol ne vom ocupa de următoarele categorii de module păstrate în dicţionarul de date:

- funcţii stocate folosite în interogări SQL;
- valori implicite ale unor atribute;
- reguli de validare complexe (la nivel de atribut şi de linie);
- declanşatoare (triggere).

Dacă prima categorie va constitui subiectul următoarelor rânduri/pagini, fiecăreia din restul categoriilor îi va fi asociat un paragraf distinct.

Funcţii stocate folosite în fraze SELECT

Începem prin a crea după proceduri stocate cărora li se „pasează" codul unui client şi care furnizează valorile facturilor şi încasărilor corespunzătoare clientului respectiv – vezi figura 6.8.

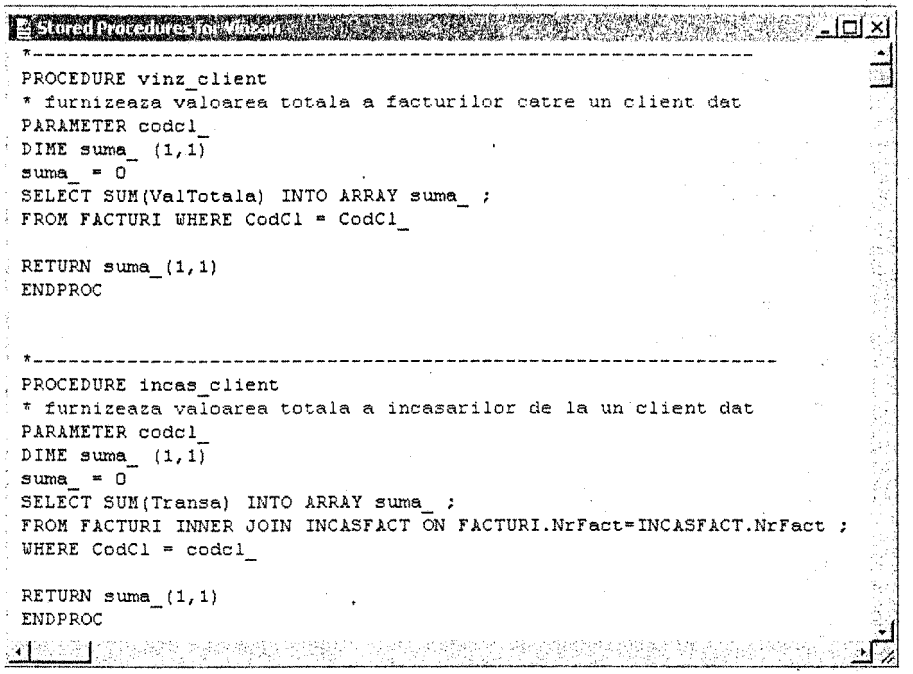
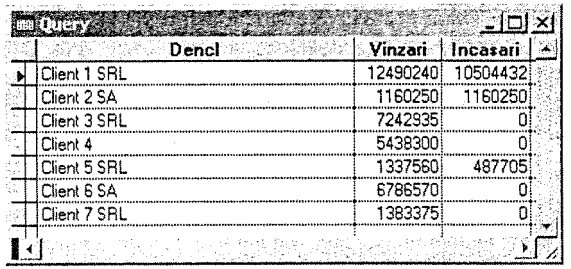


Figura 6.8. Două proceduri (funcţii) stocate

Aceste două funcţii pot fi folosite în orice consultare de o manieră asemănătoare oricărei funcţii-sistem. Astfel, pentru a obţine, pentru fiecare client, valoarea facturată şi încasată, se poate folosi o interogare de genul:

```
SELECT Dencl, vinz_client(CodCl) AS Vinzari,;
       incas_client(CodCl) AS Incasari ;
FROM CLIENTI
```

obţinuându-se un rezultat de forma celui din figura 6.9.



Dencl	Vinzari	Incasari
Client 1 SRL	12490240	10504432
Client 2 SA	1160250	1160250
Client 3 SRL	7242935	0
Client 4	5438300	0
Client 5 SRL	1337560	487705
Client 6 SA	6786570	0
Client 7 SRL	1383375	0

Figura 6.9. Rezultatul interogării care foloseşte cele două funcţii stocate

Un alt exemplu de funcție stocată, util în anumite circumstanțe (reguli de validare, declanşatoare), este prezentat în listingul 6.4. Această funcție preia codul unui produs şi returnează procentul de TVA al produsului respectiv.

Listing 6.4. Funcția stocată `AFLA_TVA`

```
PROCEDURE afla_TVA
* furnizeaza procentul TVA al unui produs dat
PARAMETER codpr_
LOCAL procent_, adresa_
procent_ = 0.00
IF !USED('produse') OR EOF('produse') OR BOF('produse')
    adresa_ = 0
    IF !USED('produse')
        USE produse IN 0
    ENDIF
ELSE
    adresa_ = RECNO('produse')
ENDIF

IF SEEK (codpr_, 'produse', 'codpr')
    procent_ = PRODUSE.ProcTVA
ENDIF

IF adresa_ > 0
    GO adresa_ IN PRODUSE
ENDIF
RETURN procent_

ENDPROC
```

La folosirea acestei funcții trebuie avut în vedere că pointerul tabelei PRODUSE se deplasează, ceea ce poate avea consecințe negative dacă apelarea sa se face în timpul editării unei înregistrări a acestei tabele în regim de buffering. După cum se observă, funcția este destul de „precaută", în sensul că, după căutare, repoziționează pointerul tabelei pe înregistrarea pe care l-a „găsit". În plus, nu modifică zona de lucru curentă acționând „de la distanță".

Ulterior creării funcției, atunci când se va dori aflarea procentului TVA al unui produs, se apelează funcția:

```
? afla_TVA (5)
```

De o manieră asemănătoare pot fi create funcții pentru obținerea de informații solicitate frecvent. Nu trebuie însă exagerat, deoarece aceste funcții pot fi plasate în corpul programelor „obişnuite", containerul bazei fiind astfel descongestionat. Dacă ar fi să încercăm un top al procedurilor stocate, din punctul de vedere al importanței, credem că această tipologie ar fi pe unul din ultimele locuri.

6.3. Valori implicite obținute prin funcții stocate

O facilitate care nu se regăsește chiar la multe dintre SGBD-urile din lumea bună a bazelor de date o constituie calcularea valorilor implicite pentru unele dintre atribute prin intermediul funcțiilor stocate. Există suficiente situații în care cheia primară a unei tabele nu poate fi „încredințată" unui atribut sau unei combinații de atribute. Soluția o constituie

cheile-surogat care nu au o semnificație anume, ci se acordă automat, de obicei prin incrementare.

Să luăm cazul tabelei INCASARI. Cheia primară a acestei tabele este `CodInc` – un caz tipic de cheie-surogat. Pentru ca la adăugarea unei noi înregistrări în această tabelă valoarea atributului `CodInc` să fie mai mare cu 1 decât cea mare mare existentă în tabelă la momentul adăugării, se poate defini o funcție stocată specială, denumită `def_codinc_incasari`.

Listing 6.5. Funcția care asigură valorile implicite ale `INCASARI.CodInc`

```
*****************************************
PROCEDURE def_codinc_incasari
*****************************************
LOCAL codinc_
DIME codinc_(1,1)
codinc_ = 0
SELECT MAX(CodInc) FROM incasari INTO ARRAY codinc_
IF codinc_(1,1) = 0
    RETURN 10001
ELSE
    RETURN codinc(1,1) + 1
ENDIF
ENDPROC
```

La prima linie inserată, valoarea returnată de funcție va fi 10001. Funcția prezentată nu este operațională decât după ce este declarată valoare implicită pentru atributul `INCASARI.CodInc`, lucru realizabil fie în mod asistat (figura 6.10), fie prin comanda:

```
ALTER TABLE INCASARI ALTER COLUMN CodInc ;
      SET DEFAULT def_codinc_incasari();
```

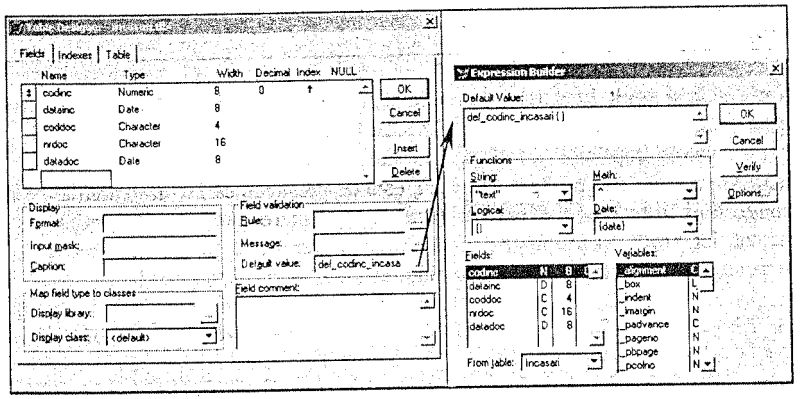


Figura 6.10. Declararea unei proceduri stocate ca valoare implicită

Atenție! Procedura stocată trebuie apelată ca şi o funcție fără parametri (cu cele două paranteze „()"), altfel este considerată nume de variabilă şi va genera o eroare: „*Variabile ... not found*".

Spre deosebire de alte SGBD-uri, în VFP, funcțiile ce întorc valori implicite pot fi oricât de sofisticate. Lucrul acesta nu trebuie exagerat, deoarece poate afecta viteza de lucru a aplicației. Ne fixăm ca obiectiv crearea unei funcții stocate care să determine cel mai frecvent client, adică acel client care apare pe cele mai multe facturi, iar codul acestuia să constituie valoarea implicită pentru atributul `CodCl` din tabela FACTURI:

```
ALTER TABLE FACTURI ALTER COLUMN CodCl
      SET DEFAULT def_codcl_facturi()
```

În listingul 6.6 se află corpul funcției (în procedurile stocate ale bazei de date):

Listing 6.6. Corpul procedurii stocate `DEF_CODCL_FACTURI`

```
*************************************
PROCEDURE def_codcl_facturi
*************************************
LOCAL vCodCl
DIME vCodCl(1,2)
vCodCl = 0

SELECT TOP 1 CodCl, COUNT(*) AS nr ;
FROM FACTURI ;
INTO ARRAY vCodCl ;
GROUP BY CodCl ;
ORDER BY nr DESC

IF vCodCl(1,1) = 0
    MESSAGEBOX('Nu se pot introduce facturi, '+CHR(13)+;
        'cita vreme nu exista nici un client !')
    RETURN .F.
ELSE
    RETURN vCodCl(1,1)
ENDIF
ENDPROC
```

Intrând și mai mult în detalii, luăm în discuție tabela LINIIFACT. Cheia primară a acesteia este combinația `(NrFact, Linie)`. Trei atribute sunt susceptibile de a se procopsi cu valori implicite calculate prin funcții stocate: `NrFact`, `Linie` și `CodPr`. Logica acestora este diferită. Spre exemplu, este de presupus că factura curentă, pentru care se introduc linii, este cea mai recentă (are cel mai mare număr). Numărul liniei trebuie incrementat, iar produsul propus de funcția stocată trebuie să fie cel mai frecvent facturat, însă nu trebuie să violeze unicitatea combinației `(NrFact, CodPr)`!

```
ALTER TABLE LINIIFACT ALTER COLUMN NrFact SET DEFAULT
def_nrfact_liniifact()

ALTER TABLE LINIIFACT ALTER COLUMN Linie SET DEFAULT
def_linie_liniifact()

ALTER TABLE LINIIFACT ALTER COLUMN CodPr SET DEFAULT
def_codpr_liniifact()
```


Listingul 6.7 prezintă codul acestor trei funcții stocate:

Listing 6.7. Corpurile procedurilor stocate `DEF_NRFACT_LINIIFACT`, `DEF_LINIE_LINIIFACT` și `DEF_CODPR_LINIIFACT`

```
*************************************
PROCEDURE def_nrfact_liniifact
*************************************
LOCAL vNrFact_
DIME vNrFact_(1)
vNrFact_ = 0

* se selecteaza cea mai recenta factura
SELECT MAX(NrFact) ;
FROM FACTURI ;
INTO ARRAY vNrFact_

IF vNrFact_(1,1) = 0
    MESSAGEBOX('Nu se pot introduce linii daca nu exista nici o factura !')
    RETURN .F.
ELSE
    RETURN vNrFact_
ENDIF
ENDPROC


*************************************
PROCEDURE def_linie_liniifact
*************************************
LOCAL vLinie_, NrFact_
DIME vLinie_(1,1)
vLinie_ = 0
NrFact_ = LINIIFACT.NrFact

* se extrage ultima linie introdusa in factura curenta
SELECT MAX(Linie) ;
FROM LINIIFACT ;
INTO ARRAY vLinie_ ;
WHERE NrFact = NrFact_

IF vLinie_(1,1) = 0                    && e prima linie din factura
    RETURN 1
ELSE                                   && se incrementeaza
    RETURN vLinie_(1,1) + 1
ENDIF
ENDPROC


*************************************
PROCEDURE def_codpr_liniifact
*************************************
LOCAL vCodPr_, NrFact_
DIME vCodPr_(1,2)
vCodPr_ = 0
NrFact_ = LINIIFACT.NrFact

** se selecteaza cel mai frecvent produs, exceptindu-le pe cele care
**    ar viola unicitatea combinatiei (NrFact, CodPr)
```

```
SELECT TOP 1 CodPr, COUNT(*) AS nr ;
FROM LINIIFACT ;
INTO ARRAY vCodPr_ ;
WHERE CodPr NOT IN ;
    (SELECT CodPr ;
     FROM LINIIFACT ;
     WHERE NrFact = NrFact_ ) ;
GROUP BY CodPr ;
ORDER BY nr DESC

IF vCodPr_(1,1) = 0
    MESSAGEBOX('Nu se mai pot introduce produse pe aceasta factura !')
    RETURN .F.
ELSE
    RETURN vCodPr_(1,1)
ENDIF
ENDPROC
```

Înainte de a încheia, se cuvine de remarcat că, la dimensiuni foarte mari ale tabelelor FACTURI şi LINIIFACT, aceste proceduri stocate pot afecta viteza aplicaţiei. Drept care, în aceste situaţii se pot încerca optimizări bazate pe variabile de memorie (masive) – de exemplu, la deschiderea aplicaţiei, să se execute interogările şi să se stocheze în variabile speciale publice. Variantele consacrate se bazează însă pe denormalizare, adică introducerea unor atribute redundante în schema bazei, atribute care să elimine sau să reducă necesitatea unor căutări sau joncţiuni mari consumatoare de resurse.

6.4. Reguli de validare avansate

Ca şi în cazul valorilor implicite, ca regulă de validare la nivel de atribut sau la nivel de înregistrare se poate preciza numele unei funcţii care va fi evaluată; dacă rezultatul este adevărat (`.T.`), modificările efectuate vor fi acceptate, iar în caz contraz vor fi respinse.

În general, restricţiile mai complexe se introduc în declanşatoare, şi mai puţin în regulile de validare. Momentul activării regulii (verificării conformităţii modificărilor cu restricţiile declarate) este diferit:

- o regulă instituită la nivel de atribut (câmp) este activată la inserarea unei linii noi în tabelă sau la modificarea valorii atributului respectiv; activarea are loc indiferent de tipul de buffering ales, înaintea celorlalte tipuri de restricţii (la nivel de linie şi declanşatoare);
- o regulă instituită la nivel de înregistrare (linie) este activată la inserarea sau modificarea înregistrării, momentul activării plasându-se după cele ale validărilor la nivel de atribute şi înainte de declanşatoare; activarea are loc indiferent de tipul de buffering;
- în VFP 6, declanşatoarele sunt module de cod lansate automat la inserarea, modificarea sau ştergerea unei înregistrări; în afară de momentul activării, care este posterior regulilor la nivel de câmp şi de înregistrare, declanşatoarele se mai particularizează şi prin aceea că, în cazul bufferingului, nu se declanşează decât printr-o comandă `TABLEUPDATE()` sau la scrierea automată a buffer-ului (la deplasarea pe altă înregistrare).

Exemplul pe care îl vom discuta este unul de deturnare a unei funcţii de validare de la destinaţia ei „normală" către calculul automat al valorii unui alt atribut. Iată, în listingul 6.8, funcţia stocată `FVR_CANT_PRET_LINIIFACT()`, care actualizează valoarea atributului `ValCuTVA` de pe înregistrarea curentă a tabelei LINIIFACT.

Listing 6.8. Funcţia-regulă de validare pentru atributele `LINIIFACT.Cantitate` şi `LINIIFACT.PretUnit`

```
*****************************************
PROCEDURE fvr_cant_pret_liniifact
*****************************************
REPLACE LINIIFACT.ValCuTVA WITH LINIIFACT.Cantitate * LINIIFACT.PretUnit * ;
    ( 1 + afla_tva(LINIIFACT.CodPr))
RETURN .T.
ENDPROC
*****************************************
```

Funcţia ce joacă rol de regulă de validare trebuie instituită pentru ambele atribute:

```
ALTER TABLE liniifact ALTER COLUMN Cantitate SET CHECK
   fvr_cant_pret_liniifact()
şi
ALTER TABLE liniifact ALTER COLUMN PretUnit SET CHECK
   fvr_cant_pret_liniifact()
```

Regula de validare poate fi introdusă şi direct în structura tabelei, folosind `Table Designer` în zona `Field Validation` (analog funcţiilor-valoare implicită, figura 6.10).

Orice modificare a unuia dintre aceste două atribute atrage după sine execuţia funcţiei. Cel mai mare risc al acesteia pare a fi calcularea valorii cu TVA chiar în condiţiile în care codul produsului este eronat. Precauţi însă, am prevăzut, în această situaţie, ca valoarea returnată de funcţia `AFLA_TVA()` să fie 0. Şi, oricum, eroarea nu ar scăpa de declanşatorul care asigură respectarea restricţiilor referenţiale.

Ca regulă generală, se recomandă evitarea, în regulile de validare şi în declanşatoare, a folosirii comenzilor de actualizare ale tabelei curente, pentru a preveni apelul recursiv al acestor proceduri/funcţii. Contraexemplul de mai sus este însă grăitor, deşi în alt sens.

Cea mai mare parte a restricţiilor pot fi incluse fie în reguli la nivel de atribut/înregistrare, fie la nivel de declanşator. Sunt însă şi diferenţe, mai ales dacă sunt avute în vedere operaţiuni ce trebuie declanşate la ştergerea unor linii, ştergere ce nu „reuşeşte" să declanşeze nici una din cele două tipuri de reguli.

6.5. Declanşatoare

Declanşatoarele (*trigger*, în original) reprezintă un tip deosebit de proceduri stocate ale unei baze de date. Particularitatea lor esenţială ţine de faptul că, o dată create şi stocate în schema bazei, acestea sunt executate automat la operaţiuni de inserare, modificare sau ştergere ale liniilor din tabela pentru care au fost definite. La această familie a declanşatoarelor, unele SGBD-uri au mai adăugat şi alte tipuri, asociate – spre exemplu, actualizarea tabelelor derivate, conectarea unei aplicaţii-utilizator, deschiderea şi închiderea instanţei bazei de date etc.

Cele mai importante avantaje ale declanşatoarelor sunt:

- permit instituirea unor reguli de validare cu mult mai complexe decât ceea ce este posibil prin clauza CHECK;
- pot ameliora sensibil mecanismul de securitate al bazei;
- permit instituirea restricţiilor referenţiale şi implementarea diferitelor moduri de tratare a modificărilor ce pot cauza probleme de integritate referenţială;
- constituie suportul calculării automate a valorilor unor atribute.

În VFP 6 există trei tipuri de declanşatoare: pentru inserare, modificare şi ştergere de linii. Momentul declanşării lor depinde esenţial de tipul de *buffering* ales, o chestiune specifică VFP. În cele ce urmează, presupunem că opţiunea de buffering este dezactivată.

6.5.1. Declanşatoare pentru restricţii referenţiale

Mecanismul de asigurare a restricţiilor referenţiale se poate implementa grafic cu ajutorul modulului Referential Integrity Builder. Paradoxal, deşi există clauza FOREIGN KEY, aceasta instituie în VFP numai legături permanente între tabele. De aceea, la crearea prin program a bazei de date sunt necesare declanşatoare „manuale" care, totuşi, au avantajul că permit afişarea unor mesaje de eroare „în clar" sau unor restricţii şi explicaţii suplimentare.

Să începem cu tabela JUDETE. Ştergerea unui judeţ poate aduce necazuri dacă există localităţi corespunzătoare. Aşa încât este necesar un declanşator de ştergere care să verifice dacă eliminarea înregistrării poate avea loc sau nu. Comanda de creare a declanşatorului este:

```
CREATE TRIGGER ON judete FOR DELETE AS trg_del_judete()
```

Figura 6.11 ilustrează o altă modalitate de a declara un *trigger*, utilizând Table Designer.

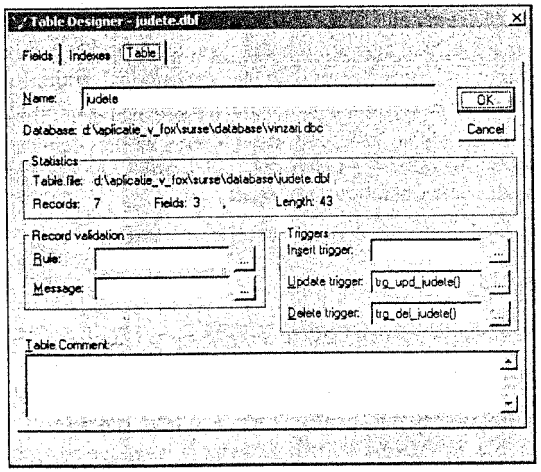


Figura 6.11. O altă modalitate de a declara declanşatoarele unei tabele

Iată, în listingul 6.9, corpul declanşatorului. Funcţia DELETED() asigură ignorarea localităţilor marcate pentru ştergere.

Listing 6.9. Varianta 1 a declanşatorului TRG_DEL_JUDETE

```
*************************************
PROCEDURE trg_del_judete
*************************************
LOCAL jud_, v
dime v(1,1)
v = ''
jud_ = judete.jud

SELECT MIN(loc) FROM localitati INTO ARRAY v WHERE jud = jud_ AND !DELETED()
IF v(1,1) # ''
    MESSAGEBOX('Exista cel putin o linie copil in tabela LOCALITATI.'+;
        CHR(13)+'Exemplu: loc. '+ALLTRIM(v(1,1)))
    RETURN .F.
ENDIF
ENDPROC
*************************************
```

SQL beneficiază de câteva optimizări ale motorului bazei date. Cu toate acestea, SELECT-ul din declanşator parcurge întreaga tabelă LOCALITATI, aşa încât pare mai rapidă varianta următoare – ce-i drept, mai puţin SQL-istă.

Listing 6.10. Varianta 2 a declanşatorului TRG_DEL_JUDETE

```
*************************************
PROCEDURE trg_del_judete
*************************************
LOCAL jud_, set_delete_
jud_ = judete.jud
set_deleted_ = SET('DELETE')
SET DELETE ON
IF INDEXSEEK(jud_, .F., 'localitati', 'jud')
    SET DELETE &set_deleted_
    MESSAGEBOX('Exista cel putin o linie copil in tabela LOCALITATI !')
    RETURN .F.
ENDIF
SET DELETE &set_deleted_
ENDPROC
*************************************
```

Funcţia INDEXSEEK() caută în tabela LOCALITATI, fără a deplasa pointerul înregistrărilor, prima linie pentru care cheia indexului JUD (cheie care este atributul cu acelaşi nume) are aceeaşi valoare ca a variabilei jud_. Când căutarea este încununată de succes, valoarea returnată este TRUE (.T. în notaţia VFP), altminteri, FALSE. Pentru ca eventualele înregistrări marcate pentru ştergere să nu fie luate în considerare de funcţia de căutare, este necesar ca SET DELETE să fie ON. Pentru igiena aplicaţiei, înainte de folosirea funcţiei, se stochează starea curentă a SET DELETE, care va fi restaurată la ieşirea din declanşator.

Modificarea indicativului unui judeţ trebuie să se propage şi în tabela LOCALITATI, ocazie bună pentru încă un declanşator special:

```
CREATE TRIGGER ON judete FOR UPDATE AS trg_upd_judete()
```

Corpul declanşatorului este prezentat în listingul 6.11. Valoarea dinaintea modificării se obţine în VFP prin funcţia OLDVAL().

Listing 6.11. Declanşator pentru modificarea unei linii din tabela JUDETE

```
*************************************
PROCEDURE trg_upd_judete
*************************************
LOCAL jud_OLD, jud_NEW
jud_NEW = judete.jud
jud_OLD = OLDVAL('jud', 'judete')
IF jud_NEW # jud_OLD
   UPDATE localitati SET jud = jud_NEW WHERE jud = jud_OLD
ENDIF
ENDPROC
```

La inserarea unei noi linii într-o tabelă-copil sau la schimbarea valorii unei chei străine trebuie verificat dacă noile valori se regăsesc în tabela-părinte. Spre exemplu, dacă în tabela LOCALITATI se modifică valoarea atributului Jud, această nouă valoare trebuie căutată în JUDETE.

```
CREATE TRIGGER ON localitati FOR UPDATE
      AS trg_upd_localitati()
```

Trebuie însă să avem în vedere că modificarea atributului Jud din LOCALITATI poate fi şi consecinţa declanşatorului trg_upd_judete. Aşa încât, pentru a evita circularitatea, se verifică dacă modificarea este una locală sau provine din declanşatorul tabelei-părinte – vezi listingul 6.12.

Listing 6.12. Declanşator TRG_UPD_LOCALITATI

```
*************************************
PROCEDURE trg_upd_localitati
*************************************
LOCAL jud_OLD, jud_NEW, codpost_OLD, codpost_NEW, v ;
DIME v(1,1)

*** mai intii se verifica restrictia referentiala cu tabela parinte
jud_NEW = localitati.jud
jud_OLD = OLDVAL('jud', 'localitati')

IF jud_NEW # jud_OLD
   IF _TRIGGERLEVEL = 1                    && modificarea este locala (nu vine din alt trigger)
      SELECT jud FROM judete INTO ARRAY v WHERE jud = jud_NEW
      IF _TALLY = 0
            MESSAGEBOX('Noua valoarea a atributului Jud - '+jud_NEW+CHR(13)+;
                  ' nu se regaseste in tabela parinte JUDETE !')
            RETURN .F.
      ENDIF
ENDIF
ENDIF

*** se propaga eventuala modificare a CodPost in tabela copil - CLIENTI
codpost_NEW = localitati.codpost
codpost_OLD = OLDVAL('codpost', 'localitati')
```



```
IF codpost_NEW # codpost_OLD
   UPDATE clienti SET codpost = codpost_NEW WHERE codpost = codpost_OLD
ENDIF
ENDPROC
```

Variabila sistem _TRIGGERLEVEL indică tocmai dacă avem de-a face cu o modificare în cascadă sau una locală a atributului Jud. Ca un alt element de noutate, funcţia sistem _TALLY întoarce numărul de linii extrase prin clauza WHERE a ultimei comenzi SQL (SELECT). Pentru a accelera căutarea în tabela-părinte, se poate folosi, în locul SELECT-ului, funcţia INDEXSEEK.

6.5.2. Atribute actualizate prin declanşatoare

Funcţia stocată declarată ca regulă de validare pentru atributele LINIIFACT.Cantitate şi LINIIFACT.PretUnit rezolvă cu brio problema actualizării câmpului LINIIFACT.ValCuTVA la modificări ale celor două atribute. Necazul este că o comandă de inserare de genul:

```
INSERT INTO LINIIFACT VALUES (23456, 3, 5, 1750, 18560, 0)
```

conduce la o valoare eronată a atributului ValCuTVa, care va fi, după această comandă, 0. Am fi astfel tentaţi să introducem un apel al funcţiei-regulă de validare de către declanşatorul de inserare al tabelei LINIIFACT, ca în listingul 6.13.

Listing 6.13. Declanşator TRG_INS_LINIIFACT

```
*************************************
PROCEDURE trg_ins_liniifact
*************************************

*** se verifica restrictia referentiala cu tabela parinte FACTURI
IF liniifact.nrfact # facturi.nrfact AND !INDEXSEEK(liniifact.nrfact, .F., 'facturi', 'nrfact')
   MESSAGEBOX('Noua valoare a atributului NrFact -';
      +LTRIM(STR(liniifact.nrfact,8)) + ;
      CHR(13)+'nu exista in tabela parinte FACTURI !')
   RETURN .F.
ENDIF

*** se verifica restrictia referentiala cu tabela parinte PRODUSE
IF liniifact.codpr # produse.codpr AND !INDEXSEEK(liniifact.codpr, .F., 'produse', 'codpr')
   MESSAGEBOX('Noua valoare a atributului CodPr -';
      +LTRIM(STR(liniifact.codpr,6)) + ;
      CHR(13)+'nu exista in tabela parinte PRODUSE !')
   RETURN .F.
ENDIF

* daca se forteaza calcularea valorii LINIIFACT.ValCuTVA
* fvr_cant_pret_liniifact()
* va aparea mesajul de eroare CANNOT UPDATE THE CURSOR

* se actualizeaza FACTURI.ValTotala
UPDATE facturi SET valtotala = valtotala + liniifact.valcutva ;
      WHERE nrfact = liniifact.nrfact

ENDPROC
```

Atenţie! **Într-un declanşator este interzisă modificarea vreunei linii din tabela pentru care a fost definit declanşatorul respectiv.** Cu toate acestea, lucrurile pot fi rezolvate lejer. Funcţia-regulă de validare nu este „bruiată" de declanşator dacă în comanda `INSERT` nu facem referire la atributul `ValCuTVA`. Rezultă că, obligatoriu, forma comenzii de adăugare este:

```
INSERT INTO LINIIFACT (NrFact, Linie, CodPr, ;
    Cantitate, PretUnit) VALUES (23456, 3, 5, 1750, 18560)
```

La preluarea valorilor atributelor `Cantitate` şi `PretUnit`, procedura se declanşează fără a crea probleme *trigger*-ului.

Ar mai fi de discutat un artificiu prezent în declanşator. Este posibil ca în aplicaţiile în care se lucrează cu buffering (specific lucrului în reţea, când mai mulţi utilizatori partajează aceeaşi tabelă), înregistrarea-părinte din FACTURI (sau chiar PRODUSE) să nu fi fost „comisă" din buffer pe disc. În acest caz, funcţia `INDEXSEEK()` întoarce `FALSE` chiar dacă valoarea-copil este corectă. De aceea, condiţia ce foloseşte `INDEXSEEK()` este precedată de o alta mai simplă, care verifică dacă factura (şi produsul) curent din tabela-părinte nu este chiar cea pentru care se introduc înregistrări în LINIIFACT.

Următorul declanşator pe care îl vom discuta este cel corespunzător actualizării fiecărei înregistrări din tabela LINIIFACT – listingul 6.14.

```
CREATE TRIGGER ON liniifact FOR UPDATE AS trg_upd_liniifact()
```

Acesta are mai multe sarcini:

- verifică dacă s-a produs o modificare a atributului `NrFact` şi dacă eventuala valoare nouă se regăseşte în tabela FACTURI;
- dacă s-a modificat `CodPr`, se testează existenţa noii valori în PRODUSE;
- dacă s-a modificat `ValCuTVA`, se actualizează valoarea atributului `ValTotala` în tabela FACTURI.

Listing 6.14. Declanşatorul de modificare a unei înregistrări din `LINIIFACT`

```
*****************************************
PROCEDURE trg_upd_liniifact
*****************************************

LOCAL nrfact_OLD, nrfact_NEW, codpr_OLD, codpr_NEW, v, ;
    valcutva_OLD, valcutva_NEW

valcutva_OLD = OLDVAL('valcutva', 'liniifact')
valcutva_NEW = liniifact.valcutva

DIME v(1,1)

set_deleted_ = SET('DELETE')
SET DELETE ON

*** se verifica restrictia referentiala cu tabela parinte FACTURI
nrfact_NEW = liniifact.nrfact
nrfact_OLD = OLDVAL('nrfact', 'liniifact')

IF nrfact_NEW # nrfact_OLD
    IF _TRIGGERLEVEL <= 1                    && modificarea este locala
        IF !USED('facturi')
            USE facturi IN 0
        ENDIF
        IF nrfact_NEW # facturi.nrfact AND !INDEXSEEK(nrfact_NEW, .F., 'facturi', 'nrfact')
            MESSAGEBOX('Noua valoare a atributului NrFact -' ;
                +LTRIM(STR(nrfact_NEW,8)) + CHR(13)+;
                'nu exista in tabela parinte FACTURI !')
            SET DELETE &set_deleted_
            RETURN .F.
        ENDIF
    ENDIF
ENDIF


*** se verifica restrictia referentiala cu tabela parinte PRODUSE
codpr_NEW = liniifact.codpr
codpr_OLD = OLDVAL('codpr', 'liniifact')

IF codpr_NEW # codpr_OLD
    IF _TRIGGERLEVEL <= 1                    && modificarea este locala
        IF !USED('produse')
            USE produse IN 0
        ENDIF
        IF codpr_NEW # produse.codpr AND !INDEXSEEK(codpr_NEW, .F., 'produse', 'codpr')
            MESSAGEBOX('Noua valoare a atributului CodPr -'+;
LTRIM(STR(codpr_NEW,6)) + ;
                CHR(13)+'nu exista in tabela parinte PRODUSE !')
            SET DELETE &set_deleted_
            RETURN .F.
        ENDIF
    ENDIF
ENDIF


*** Daca s-a schimbat numarul facturii, linia trebuie scoasa de la fosta factura
** si adaugata la noua factura
IF nrfact_NEW # nrfact_OLD  AND _TRIGGERLEVEL <= 1

    * se scade valoarea de la factura "veche"
    UPDATE facturi SET valtotala = valtotala - valcutva_OLD ;
        WHERE nrfact = nrfact_OLD

    * se aduna valoarea de la factura "noua"
    UPDATE facturi SET valtotala = valtotala + valcutva_NEW ;
        WHERE nrfact = nrfact_NEW

ELSE  ** nu s-a modificat NrFact

    IF valcutva_OLD # valcutva_NEW
        UPDATE facturi SET valtotala = valtotala -  valcutva_OLD + valcutva_NEW ;
            WHERE nrfact = nrfact_OLD
    ENDIF
ENDIF
SET DELETE &set_deleted_
ENDPROC
*****************************************
```


La ştergerea oricărei linii dintr-o tabelă-copil nu există nici un risc cu privire la integritatea referenţială a bazei. Totuşi, ne putem propune să construim un declanşator

pentru ştergerea unei înregistrări din LINIIFACT care să verifice dacă înregistrarea ştearsă este ultima pentru factura respectivă, caz în care să se şteargă şi linia-părinte din FACTURI. În plus, înregistrarea ştearsă trebuie neapărat să reprezinte ultima linie din factură.

```
CREATE TRIGGER ON liniifact FOR DELETE AS trg_del_liniifact()
```

Corpul declanşatorului se află în listingul 6.15.

Listing 6.15. Declanşatorul de ştergere a unei înregistrări din LINIIFACT

```
**************************************
PROCEDURE trg_del_liniifact
**************************************
LOCAL cite_, NrFact_, linie_
DIME cite_(1,1), linie_(1,1)
cite_ = 0
linie_ = 0
NrFact_ = liniifact.nrfact

* se interzice stergerea altei linii decit ultima
SELECT MAX(Linie) INTO ARRAY linie_ FROM LINIIFACT ;
WHERE NrFact = NrFact_ AND !DELETED()

SELECT liniifact

IF linie_(1,1) # liniifact.linie
    MESSAGEBOX('Nu puteti sterge decit ultima linie din factura !')
    RETURN .F.
ENDIF

* se verifica daca mai ramine vreo linie in factura
SELECT COUNT(*) INTO ARRAY cite_ FROM LINIIFACT ;
WHERE NrFact = NrFact_ AND !DELETED()

IF cite_(1,1) = 1
    MESSAGEBOX('Aceasta a fost ultima linie din factura !')
    IF SEEK(NrFact_, 'facturi', 'nrfact')
        SELECT facturi
        DELETE
    ENDIF
ELSE
    * se actualizeaza FACTURI.ValTotala
    UPDATE facturi SET valtotala = valtotala - liniifact.valcutva ;
        WHERE nrfact = liniifact.nrfact
ENDIF
SELECT liniifact
RETURN .T.
ENDPROC
**************************************
```

6.5.3. Declanşatoare pentru controlul actualizărilor

În materie de securitate, VFP nu stă grozav. Cu toate acestea, pot fi valorificate atuurile declanşatoarelor şi ale sistemului de operare Windows NT/2000 şi se pot institui mecanisme acceptabile pentru siguranţa exploatării aplicaţiilor. Dar acesta va fi subiectul unui alt capitol.

Deocamdată ne propunem ca unele atribute, precum cele calculate, să nu fie modificate direct, prin comenzi BROWSE, EDIT sau REPLACE, sau prin legarea direct la controalele unui formular, ci numai prin declanşatoare sau reguli-funcţii de validare. Pentru a permite modificarea valorii unui atribut numai prin intermediul unui declanşator, la actualizare se verifică dacă nivelul declanşatorului este mai mare decât 1, ceea ce ar constitui un indiciu că modificarea provine din alt *trigger*, şi se validează operaţiunea. Dacă funcţia _TRIGGERLEVEL întoarce o valoare egală cu 1, modificarea este respinsă. Să exemplificăm cu declanşatorul pentru actualizarea tabelei FACTURI.

```
CREATE TRIGGER ON facturi FOR UPDATE AS trg_upd_facturi()
```

Corpul declanşatorului este prezentat în listingul 6.16.

Listing 6.16. Declanşatorul pentru modificarea unei linii din tabela FACTURI

```
**************************************
PROCEDURE trg_upd_facturi
**************************************

LOCAL codcl_OLD, codcl_NEW, valtotala_OLD, valtotala_NEW, ;
    nrfact_OLD, nrfact_NEW, gestiune_OLD, gestiune_NEW

set_deleted_ = SET('DELETE')
SET DELETE ON

*** se verifica restrictia referentiala cu tabela parinte GESTIUNI
gestiune_NEW = facturi.gestiune
gestiune_OLD = OLDVAL('gestiune', 'facturi')
IF gestiune_NEW # gestiune_OLD
    IF _TRIGGERLEVEL = 1                    && modificarea este locala
        IF !USED('gestiuni')
            USE gestiuni IN 0
        ENDIF
        IF gestiune_NEW # gestiuni.gestiune AND ;
!INDEXSEEK(gestiune_NEW, .F., 'gestiuni', 'gestiune')
            MESSAGEBOX('Noua valoare a atributului Gestiune -';
                +LTRIM(gestiune_NEW) + ;
                CHR(13)+'nu exista in tabela parinte GESTIUNI !')
            SET DELETE &set_deleted_
            RETURN .F.
        ENDIF
    ENDIF
ENDIF

*** se verifica restrictia referentiala cu tabela parinte CLIENTI
codcl_NEW = facturi.codcl
codcl_OLD = OLDVAL('codcl', 'facturi')

IF codcl_NEW # codcl_OLD
    IF _TRIGGERLEVEL <= 1                   && modificarea este locala
        IF !USED('clienti')
            USE clienti IN 0
        ENDIF
        IF codcl_NEW # clienti.codcl AND !INDEXSEEK(codcl_NEW, .F., 'clienti', 'codcl')
            MESSAGEBOX('Noua valoare a atributului CodCl -';
                +LTRIM(STR(codcl_NEW,6)) + ;
                CHR(13)+'nu exista in tabela parinte CLIENTI !')
            SET DELETE &set_deleted_
```

```
                RETURN .F.
            ENDIF
        ENDIF
    ENDIF


    *** se interzice modificarea interactiva a atributului ValTotala
    valtotala_NEW = facturi.valtotala
    valtotala_OLD = OLDVAL('valtotala', 'facturi')
    IF valtotala_NEW # valtotala_OLD
        IF _TRIGGERLEVEL = 1
            MESSAGEBOX('Nu puteti modifica interactiv valoarea totala a facturii !')
            SET DELETE &set_deleted_
            RETURN .F.
        ENDIF
    ENDIF

    *** se propaga eventuala modificare a NrFact in tabelele copil - LIINIIFACT si INCASFACT
    nrfact_NEW = facturi.nrfact
    nrfact_OLD = OLDVAL('nrfact', 'facturi')
    IF nrfact_NEW # nrfact_OLD
        UPDATE liniifact SET NrFact = nrfact_NEW WHERE NrFact = nrfact_OLD
        UPDATE incasfact SET NrFact = nrfact_NEW WHERE NrFact = nrfact_OLD
    ENDIF

    SET DELETE &set_deleted_
    ENDPROC
    ***************************************
```

În schimb, declanşatorul de inserare trebuie să verifice ca valoarea atributului să fie zero – listingul 6.17.

Listing 6.17. Declanşatorul pentru inserarea unei înregistrări în tabela `FACTURI`

```
    ***************************************
    PROCEDURE trg_ins_facturi
    ***************************************
    LOCAL codcl_OLD, codcl_NEW, gestiune_OLD, gestiune_NEW

    set_deleted_ = SET('DELETE')
    SET DELETE ON

    *** se verifica restrictia referentiala cu tabela parinte GESTIUNI
    gestiune_NEW = facturi.gestiune
    IF !USED('gestiuni')
        USE gestiuni IN 0
    ENDIF
    IF gestiune_NEW # gestiuni.gestiune AND !INDEXSEEK(gestiune_NEW, .F., 'gestiuni', 'gestiune')
            MESSAGEBOX('Noua valoare a atributului Gestiune -';
            +LTRIM(gestiune_NEW) + ;
            CHR(13)+'nu exista in tabela parinte GESTIUNI !')
        SET DELETE &set_deleted_
        RETURN .F.
    ENDIF

    *** se verifica restrictia referentiala cu tabela parinte CLIENTI
    codcl_NEW = facturi.codcl
    IF !USED('clienti')
```



```
        USE clienti IN 0
    ENDIF
    IF codcl_NEW # clienti.codcl AND !INDEXSEEK(codcl_NEW, .F., 'clienti', 'codcl')
        MESSAGEBOX('Noua valoare a atributului CodCl -';
            +LTRIM(STR(codcl_NEW,6)) + ;
            CHR(13)+'nu exista in tabela parinte CLIENTI !')
        SET DELETE &set_deleted_
        RETURN .F.
    ENDIF


    *** se interzice o valoare diferita de zero a atributului ValTotala
    IF facturi.valtotala # 0
        MESSAGEBOX('O factura noua nu poate avea decit valoarea zero'+;
            CHR(13)+'(este calculata automat ulterior)')
        SET DELETE &set_deleted_
        RETURN .F.
    ENDIF

    SET DELETE &set_deleted_
    ENDPROC
    ***************************************
```

BIBL. CENTR. UNIV. „M. EMINESCU" IAŞI INFORMATICA

În tabela LINIIFACT, atributul `ValCuTVA` este calculat prin funcţia-regulă de validare `fvr_cant_pret_liniifact()`. Aceasta se lansează la orice modificare a unuia dintre câmpurile `LINIIFACT.Cantitate` şi `LINIIFACT.PretUnit`. Nimic nu interzice însă, la acest moment, ca printr-o comandă sau formular, utilizatorii să modifice direct atributul calculat. Variabila-sistem `_TRIGGERLEVEL` nu ne este de mare folos, deoarece modificarea nu e rezultatul declanşatorului. Am fi tentaţi să folosim aceeaşi funcţie stocată – `fvr_cant_pret_liniifact()` – şi ca funcţie-regulă de validare pentru `LINIIFACT.ValCuTVA`. Nu este indicat, mult mai la îndemână fiind o nouă funcţie – vezi listingul 6.18.

```
ALTER TABLE liniifact ALTER COLUMN valcutva SET CHECK
   fvr_valcutva_liniifact() NOVALIDATE
```

Listing 6.18. Corpul procedurii/funcţiei stocate ce reprezintă regula de validare pentru `LINIIFACT.ValCuTVA`

```
    ***************************************
    PROCEDURE fvr_valcutva_liniifact
    ***************************************
    IF LINIIFACT.ValCuTVA # LINIIFACT.Cantitate * ;
        LINIIFACT.PretUnit * ( 1 + afla_tva(LINIIFACT.CodPr))
        MESSAGEBOX('Nu puteti modifica interactiv atributul ValCuTVA')
        RETURN .F.
    ENDIF
    RETURN .T.
    ENDPROC
```

Listingul tuturor procedurilor stocate (ar fi trebui să fie listingul 6.19) este mult prea lung şi nu a fost inclus în carte, dar totuşi poate fi descărcat de la adresa http://www.es.uaic.ro/infoec/books.

6.6. Salvarea/restaurarea procedurilor stocate. Refacerea containerului BD

Una dintre cele mai disperate situații în utilizarea aplicațiilor VFP este cea în care, din diverse motive, baza de date, și în primul rând containerul, este alterată. Riscurile sporesc pe măsura creșterii dimensiunii tabelelor și a numărului de utilizatori conectați simultan la aplicație (în rețea).

Pierderea dicționarului de date creează fiori oricărui responsabil de aplicație și nici un efort nu este prea mic pentru a preîntâmpina sau, în ultimă instanță, a salva situația. Din fericire, există o comandă destul de simplă – `COPY PROCEDURES` –, prin care procedurile stocate ale bazei de date pot fi salvate într-un fișier `ASCII`. Spre exemplu, comanda

```
COPY PROCEDURES TO proc_stoc_vinzari.txt
```

va crea un fișier text numit `proc_stoc_vinzari`, care va conține copia procedurilor stocate din containerul bazei. O idee bună ar fi păstrarea tuturor (sau celor mai recente) versiuni ale procedurilor stocate, caz în care numele fișierului în care are loc salvarea include și data salvării:

```
COPY PROCEDURES TO 'copie_ps_vinzari_'+DTOS(date())+'.txt'
```

À la limite, în numele fișierului se poate include și ora la care s-a făcut salvarea – dar ar fi o exagerare justificabilă în puține cazuri.

Atunci când există indicii că procedurile stocate au fost alterate sau chiar distruse, comanda de refacere a acestora prin preluarea dintr-un fișier în care au fost salvate în prealabil este `APPEND PROCEDURES FROM fișier [OVERWRITE]`:

```
APPEND PROCEDURES FROM proc_stoc_vinzari.txt OVERWRITE
```

Pentru a reface procedurile stocate prin preluarea din fișierul text cel mai recent, se poate folosi programul prezentat în listingul 6.20.

Listing 6.20. Refacerea procedurilor stocate

```
* se determina ultima varianta (salvata) a procedurilor stocate
* si se reface containerul bazei

* se preiau in masivul vFisiere toate fisierele ce contin procedurile stocate
nr = ADIR(vFisiere, 'copie_ps_vinzari_*.txt')

* se preia numai numele fiecarui fisiere intr-un vector (masiv unidimensional)
DIME vNumeFis(nr)
FOR i = 1 TO nr
    vNumeFis (i) = vFisiere(i,1)
ENDFOR

* se sorteaza vectorul
= ASORT(vNumeFis)

* cea mai recenta versiune este:
ultimul_ = vNumeFis(nr)

* se refac procedurile in containerul BD
APPEND PROCEDURES FROM (ultimul_) OVERWRITE
```


Uneori însă, simpla refacere a procedurilor stocate nu rezolvă situația de criză a unei aplicații. De obicei, primii care cedează sunt indecșii, așa că este necesar și un program de refacere a indecșilor (vezi capitolul 2). Am întâlnit însă și situații în care nici refacerea indecșilor și a procedurilor stocate nu a reușit remedierea containerului bazei. Astfel încât, în continuare, vă prezentăm o variantă care, deși brutală, reușește să repună pe picioare aproape orice bază de date, în care însă tabelele sunt „coerente" – listingul 6.21.

Listing 6.21. Refacerea bazei de date

```
*** RE-CONSTITUIREA BAZEI DE DATE - varianta radicala !

CLOSE DATA ALL
CLOSE TABLES ALL
CLEAR ALL
CLOSE ALL
SET SAFETY OFF

* se sterge BD (inclusiv dictionarul), insa nu si tabelele
DELETE DATABASE vinzari

* se sterge indexul bazei de date
DELETE FILE *.DCX

* se sterg indecsii
DELETE FILE *.CDX

* se re-creaza baza
CREATE DATABASE vinzari

* se adauga tabelele care, in acest moment, sunt independente
* ( trebuie avut grija la numele lungi ale atributelor... vor trebui redenumiti explicit !)
ADD TABLE judete
USE judete IN 0
PACK
ADD TABLE localitati
USE localitati IN 0
PACK
ADD TABLE clienti
USE clienti IN 0
PACK
ADD TABLE persoane
USE persoane IN 0
PACK
ADD TABLE persclienti
USE persclienti IN 0
PACK
ADD TABLE produse
USE produse IN 0
PACK
ADD TABLE gestiuni
USE gestiuni IN 0
PACK
ADD TABLE facturi
USE facturi IN 0
PACK
ADD TABLE liniifact
USE liniifact IN 0
```

```
PACK
ADD TABLE incasari
USE incasari IN 0
PACK
ADD TABLE incasfact
USE incasfact IN 0
PACK


* se creaza indecsii primari si candidati

ALTER TABLE judete ADD PRIMARY KEY jud TAG jud
ALTER TABLE judete ADD UNIQUE judet TAG judet

ALTER TABLE localitati ADD PRIMARY KEY codpost TAG codpost
ALTER TABLE localitati ADD FOREIGN KEY jud TAG jud REFERENCES judete TAG jud

ALTER TABLE clienti ADD PRIMARY KEY codcl TAG codcl
ALTER TABLE clienti ADD FOREIGN KEY codpost TAG codpost ;
  REFERENCES localitati TAG codpost

ALTER TABLE persoane ADD PRIMARY KEY cnp TAG cnp
ALTER TABLE persoane ADD FOREIGN KEY codpost TAG codpost ;
  REFERENCES localitati TAG codpost

ALTER TABLE persclienti ADD PRIMARY KEY cnp+STR(codcl,6)+functie TAG primaru
ALTER TABLE persclienti ADD FOREIGN KEY cnp TAG cnp REFERENCES persoane TAG cnp
ALTER TABLE persclienti ADD FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl

ALTER TABLE produse ADD PRIMARY KEY codpr TAG codpr

ALTER TABLE gestiuni ADD PRIMARY KEY gestiune TAG gestiune
ALTER TABLE gestiuni ADD FOREIGN KEY codpost TAG codpost REFERENCES localitati TAG codpost
ALTER TABLE gestiuni ADD FOREIGN KEY cnp TAG cnp REFERENCES persoane TAG cnp

ALTER TABLE facturi ADD PRIMARY KEY nrfact TAG nrfact
ALTER TABLE facturi ADD FOREIGN KEY gestiune TAG gestiune ;
  REFERENCES gestiuni TAG gestiune
ALTER TABLE facturi ADD FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl

ALTER TABLE liniifact ADD PRIMARY KEY STR(nrfact,8)+STR(linie,2) TAG primaru
ALTER TABLE liniifact ADD FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact
ALTER TABLE liniifact ADD FOREIGN KEY codpr TAG codpr REFERENCES produse TAG codpr

ALTER TABLE incasari ADD PRIMARY KEY codinc TAG codinc

ALTER TABLE incasfact ADD PRIMARY KEY STR(codinc,8)+STR(nrfact,8) TAG primaru
ALTER TABLE incasfact ADD FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact
ALTER TABLE incasfact ADD FOREIGN KEY codinc TAG codinc REFERENCES incasari TAG codinc

* se refac procedurile stocate
* APPEND PROCEDURES FROM proceduri_stocate_vinzari.TXT OVERWRITE
DO refacere_proceduri_stocate

* declararea declansatoarelor
PUBLIC vTabele
DIME vTabele(1,1)
ADBOBJECTS(vTabele, "TABLE")
FOR i = 1 TO ALEN(vTabele)
```



```
  tab  = vTabele(i)
  trigger_ = "trg_del_"+allt(tab_)+"()"
  CREATE TRIGGER ON &tab_ FOR DELETE AS &trigger_
  trigger_ = "trg_upd_"+allt(tab_)+"()"
  CREATE TRIGGER ON &tab_ FOR UPDATE AS &trigger_
  trigger_ = "trg_ins_"+allt(tab_)+"()"
  CREATE TRIGGER ON &tab_ FOR INSERT AS &trigger_
ENDFOR

* refacerea regulilor de validare si valorilor implicite
DO checkuri_nulli_si_defaulturi

SET SAFETY ON

RETURN
```

Programul de refacere a valorilor implicite şi regulilor de validare – `checkuri_nulli_si_defaulturi` – este cel din listingul 6.22.

Listing 6.22. Programul de re-declarare a valorilor implicite şi regulilor de validare

```
** reguli de validare la nivel de atribut

ALTER TABLE JUDETE ALTER COLUMN ;
  jud SET CHECK (jud=LTRIM(UPPER(jud))) ;
          ERROR 'Indicativul judetului se scrie cu majuscule !'

ALTER TABLE JUDETE ALTER COLUMN ;
  judet SET CHECK (judet=LTRIM(PROPER(judet))) ;
          ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
          'denumirii judetului este majuscula; '+CHR(13)+;
          'restul literelor sunt mici!'

ALTER TABLE JUDETE ALTER COLUMN regiune SET DEFAULT 'Moldova'

ALTER TABLE JUDETE ALTER COLUMN ;
  regiune SET CHECK (INLIST(regiune, 'Banat', 'Transilvania', ;
      'Dobrogea', 'Oltenia', 'Muntenia', 'Moldova')) ;
          ERROR 'Regiunea poate avea o valoare numai din lista:'+CHR(13)+;
          'Banat, Transilvania, Dobrogea, Oltenia, Muntenia, Moldova'


ALTER TABLE LOCALITATI ALTER COLUMN ;
  codpost SET CHECK (codpost=LTRIM(codpost)) ;
          ERROR 'Codul postal se scrie fara spatii la inceput !'

ALTER TABLE LOCALITATI ALTER COLUMN ;
  loc SET CHECK (loc=LTRIM(PROPER(loc))) ;
          ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
          'denumirii localitatii este majuscula; '+CHR(13)+;
          'restul literelor sunt mici!'

ALTER TABLE LOCALITATI ALTER COLUMN jud SET DEFAULT 'IS'


ALTER TABLE CLIENTI ALTER COLUMN ;
  codcl SET CHECK (codcl > 1000)
```

```
ALTER TABLE CLIENTI ALTER COLUMN ;
  codcl SET DEFAULT def_codcl_clienti()

ALTER TABLE CLIENTI ALTER COLUMN ;
  dencl SET  CHECK (SUBSTR(dencl,1,1) = UPPER(SUBSTR(dencl,1,1))) ;
        ERROR 'Prima litera din denumirea clientului este ; '+CHR(13)+;
        'obligatoriu majuscula !'

ALTER TABLE CLIENTI ALTER COLUMN codfiscal NULL

ALTER TABLE CLIENTI ALTER COLUMN ;
  codfiscal SET CHECK (SUBSTR(codfiscal,1,1) = UPPER(SUBSTR(codfiscal,1,1))) ;
        ERROR 'Prima (eventuala) litera din codul fiscal ; '+CHR(13)+;
        'este obligatoriu majuscula !'

ALTER TABLE CLIENTI ALTER COLUMN adresa NULL

ALTER TABLE CLIENTI ALTER COLUMN ;
  adresa SET CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
        ERROR 'Prima (eventuala) litera din adresa ; '+CHR(13)+;
        'este obligatoriu majuscula !'

ALTER TABLE CLIENTI ALTER COLUMN codpost SET DEFAULT '6600'

ALTER TABLE CLIENTI ALTER COLUMN telefon NULL

ALTER TABLE PERSOANE ALTER COLUMN ;
  cnp SET CHECK (cnp=LTRIM(UPPER(cnp))) ;
        ERROR 'Codul numeric personal se scrie fara spatii la inceput !'

ALTER TABLE PERSOANE ALTER COLUMN ;
  nume SET CHECK (nume=LTRIM(PROPER(nume))) ;
        ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
        'numelui este majuscula; '+CHR(13)+;
        'restul literelor sunt mici!'

ALTER TABLE PERSOANE ALTER COLUMN ;
  prenume SET CHECK (prenume=LTRIM(PROPER(prenume))) ;
        ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
        'prenumelui este majuscula; '+CHR(13)+;
        'restul literelor sunt mici!'

ALTER TABLE PERSOANE ALTER COLUMN adresa NULL

ALTER TABLE PERSOANE ALTER COLUMN ;
  adresa SET CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
        ERROR 'Prima litera din adresa este obligatoriu majuscula !'

ALTER TABLE PERSOANE ALTER COLUMN sex SET DEFAULT 'B'

ALTER TABLE PERSOANE ALTER COLUMN ;
  sex SET CHECK (INLIST(sex,'F','B')) ;
        ERROR 'Atributul Sex poate avea valorile F (de la Femeiesc)'+CHR(13)+;
              'sau B (de la Barbatesc) !'

ALTER TABLE PERSOANE ALTER COLUMN codpost SET DEFAULT '6600'

ALTER TABLE PERSOANE ALTER COLUMN telacasa NULL
ALTER TABLE PERSOANE ALTER COLUMN telbirou NULL
```



```
ALTER TABLE PERSOANE ALTER COLUMN telmobil  NULL
ALTER TABLE PERSOANE ALTER COLUMN email NULL

ALTER TABLE PERSCLIENTI ALTER COLUMN ;
  functie SET CHECK (SUBSTR(functie,1,1) = UPPER(SUBSTR(functie,1,1))) ;
        ERROR 'Prima litera din functie este obligatoriu majuscula !'

ALTER TABLE PRODUSE ALTER COLUMN ;
  codpr SET DEFAULT def_codpr_produse()

ALTER TABLE PRODUSE ALTER COLUMN ;
  codpr SET CHECK (codpr > 0) ;
        ERROR 'Codul produsului trebuie sa fie mai mare de 100000 !'

ALTER TABLE PRODUSE ALTER COLUMN ;
  denpr SET CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))) ;
        ERROR 'Prima litera din denumirea produsului este obligatoriu majuscula !'

ALTER TABLE PRODUSE ALTER COLUMN ;
  grupa SET CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))) ;
        ERROR 'Prima litera din grupa este obligatoriu majuscula !'

ALTER TABLE PRODUSE ALTER COLUMN procTVA SET DEFAULT .19

ALTER TABLE FACTURI ALTER COLUMN ;
  nrfact SET DEFAULT def_nrfact_facturi()

ALTER TABLE FACTURI ALTER COLUMN ;
  datafact SET DEFAULT DATE()

ALTER TABLE FACTURI ALTER COLUMN ;
  datafact SET CHECK (BETWEEN(datafact,{^2000/01/01},{^2010/12/31})) ;
        ERROR 'Baza de date functioneaza in intervalul 1 ian 2000 - 31 dec.2010 !'

ALTER TABLE FACTURI ALTER COLUMN ;
  codcl SET DEFAULT def_codcl_facturi()

ALTER TABLE facturi ALTER COLUMN gestiune SET DEFAULT def_gestiune_facturi()
NOVALIDATE

ALTER TABLE FACTURI ALTER COLUMN obs NULL

ALTER TABLE LINIIFACT ALTER COLUMN ;
  nrfact SET DEFAULT def_nrfact_liniifact()

ALTER TABLE LINIIFACT ALTER COLUMN ;
  linie SET DEFAULT def_linie_liniifact()

ALTER TABLE LINIIFACT ALTER COLUMN ;
  codpr SET DEFAULT def_codpr_liniifact()

ALTER TABLE LINIIFACT ALTER COLUMN ;
  linie NUMBER(2) SET CHECK (linie > 0) ;
        ERROR 'Atributul linie trebuie sa fie mai mare ca zero !'

ALTER TABLE LINIIFACT ALTER COLUMN ;
  cantitate SET CHECK fvr_cant_pret_liniifact()
```

```
ALTER TABLE LINIIFACT ALTER COLUMN ;
  pretunit SET CHECK fvr_cant_pret_liniifact()

ALTER TABLE INCASARI ALTER COLUMN ;
  codinc SET DEFAULT def_codinc_incasari()

ALTER TABLE INCASARI ALTER COLUMN datainc SET DEFAULT DATE()

ALTER TABLE INCASARI ALTER COLUMN ;
  datainc SET CHECK (BETWEEN(datainc,{^2000/01/01},{^2010/12/31})) ;
        ERROR 'Baza de date functioneaza in intervalul 1 ian.2000 - 31 dec.2010 !'

ALTER TABLE INCASARI ALTER COLUMN ;
   coddoc SET CHECK(coddoc=UPPER(LTRIM(coddoc))) ;
        ERROR 'Codul documentului se scrie cu majuscule !'

ALTER TABLE INCASARI ALTER COLUMN datadoc SET DEFAULT DATE() - 5

ALTER TABLE INCASARI ALTER COLUMN  ;
  datadoc SET CHECK (BETWEEN(datadoc, {^2000/01/01},{^2010/12/31})) ;
        ERROR 'Data documentului trebuie sa fie intre 1 ian.2000 si 31 dec.2010 !'
```

6.7. Declanşatoare generalizate pentru asigurarea restricţiilor referenţiale

După cum am văzut în capitolul 2 şi în cel de faţă, declararea restricţiilor referenţiale se poate face fie asistat, fie prin program, prin stabilirea, mai întâi, de legături permanente între tabele (prin indecşi) şi apoi declararea, prin `Referential Integrity Builder`, sau prin declanşatoare, a regulilor de urmat la actualizarea tabelelor-părinte şi copil în vederea prezervării integrităţii bazei. Fireşte, prima soluţie prezintă avantajul simplităţii. Varianta prin declanşatoare este însă mai potrivită, deoarece permite afişarea unor mesaje clare ce pot fi preluate în formulare, astfel încât aplicaţia este mult mai uşor de exploatat. În plus, în declanşatoare pot fi incluse şi alte elemente în afara celor ce ţin de restricţiile referenţiale.

Dezavantajul principal al lucrului prin declanşatoare, aşa cum au fost prezentate în paragrafele anterioare, ţine de o anumită redundanţă. Dacă baza are zeci de tabele, şi de obicei are, este necesar un anumit volum de muncă de uzură. Plus dimensiunea mare a dicţionarului de date.

Ca urmare, vă propunem o variantă elegantă în care declanşatoarele sunt generalizate. În containerul bazei va exista câte un declanşator de inserare, modificare şi ştergere, declanşator care preia ca parametru numele tabelei actualizate.

Atenţie! Soluţia prezentată în continuare funcţionează numai atunci când restricţiile referenţiale se stabilesc printr-un singur atribut, adică nu există nici o cheie străină compusă. În plus, indecşii simpli au acelaşi nume cu atributul de indexare în toate tabelele. De asemenea, nu sunt rezolvate restricţiile stabilite între două atribute ale aceleiaşi tabele (de exemplu, într-o bază de date pentru evidenţa personalului, în tabela PERSONAL atributul `MarcaŞef` este cheie străină, tabela-părinte fiind tot PERSONAL, iar atributul de legătură al „părintelui" este `Marca`).

Funcţiile/procedurile următoare rămân neschimbate faţă de versiunea anterioară a procedurilor stocate:

- `fvr_cant_pret_liniifact`
- `fvr_valcutva_liniifact`
- `def_codcl_clienti`
- `def_codpr_produse`
- `def_nrfact_facturi`
- `def_codcl_facturi`
- `def_nrfact_liniifact`
- `def_linie_liniifact`
- `def_codpr_liniifact`
- `def_codinc_incasari`
- `vinz_client`
- `incas_client`
- `afla_TVA`.

Începem cu o procedură nouă de deschidere a tabelelor şi iniţializare a variabilelor-tablou de memorie ce conţin informaţii despre tabelele bazei (`vTabele`), şi legăturile permanente dintre tabele – suportul restricţiilor referenţiale, astfel:

- `vRelatii` este un vector (masiv unidimensional) ce conţine câte o componentă pentru fiecare dintre relaţiile permanente stabilite între două tabele ale bazei;
- `vCopii` – tabelele-copil;
- `vParinti` – tabelele-părinte;
- `vTaguriCopil` – numele indexului elementar (*tag*-ului) al tabelei-copil prin care se stabileşte legătura cu tabela-părinte;
- `vTaguriParinte` – numele indexului elementar (*tag*-ului) al tabelei-părinte prin care se stabileşte legătura cu tabela-copil.

Toate aceste variabile sunt declarate publice, pentru a putea fi folosite în procedurile stocate. Codul-sursă al procedurii `setup` este prezentat în listingul 6.23, iar cel al procedurii de salvare automată (la ieşirea din sesiunea VFP) a procedurilor stocate ale bazei în listingul 6.24.

Listing 6.23. Procedura stocată de iniţializare a variabilelor ce conţin informaţii despre relaţiile dintre tabele

```
*====================
PROCEDURE SETUP
*--------------------------
* se deschide baza de date
IF !DBUSED('vinzari')
  OPEN DATABASE vinzari EXCLUSIVE
ENDIF

* Deschiderea automata a tabelelor BD si
* stabilirea indexului activ - varianta a 2-a
* se presupune ca baza e deja deschisa
```

```
PUBLIC vTabele
DIME vTabele(5,1)
ADBOBJECTS(vTabele, "TABLE")
FOR i = 1 TO ALEN(vTabele)
  tab_ = vTabele(i)
  * daca tabela nu este deja deschisa, se asociaza unei zone de lucru
  IF !USED((tab_))
      USE (tab_) IN 0 EXCLU
  ENDIF
  * daca exista macar un index, primul (dintre indecsi) devine principal
  SELECT (tab_)
  IF TAGCOUNT() > 0
      SET ORDER TO TAG 1
  ENDIF
  PUBLIC trg_ins_&tab_, trg_upd_&tab_, trg_del_&tab_
ENDFOR

PUBLIC vRelatii, vCopii, vParinti, vTaguriCopil, vTaguriParinte
DIME vRelatii(4,4), vCopii(1), vParinti(1), vTaguriCopil(1), vTaguriParinte(1)
ADBOBJECTS(vRelatii, 'Relation')
* prima coloana din vRelatii - tabela COPIL
* a doua coloana din vRelatii - tabela PARINTE
* a treia coloana din vRelatii - index (tag) COPIL
* a patra coloana din vRelatii - index(tag) PARINTE
DIME vCopii(ALEN(vRelatii,1)), vParinti(ALEN(vRelatii,1)), ;
  vTaguriCopil(ALEN(vRelatii,1)), vTaguriParinte(ALEN(vRelatii,1))
FOR i = 1 TO ALEN(vRelatii,1)
  vCopii(i) = vRelatii(i,1)
  vParinti(i) = vRelatii(i,2)
  vTaguriCopil(i) = vRelatii(i,3)
  vTaguriParinte(i) = vRelatii(i,4)
ENDFOR

** la fiecare iesire din VFP se salveaza o copie a procedurilor stocate
ON SHUTDOWN DO salvare_proceduri_stocate

ENDPROC
*=============================================================================
```

Listing 6.24. Procedura de salvare a variantei curente a procedurilor stocate

```
*-----------------------------------------------------------------------
PROCEDURE salvare_proceduri_stocate
*-----------------------------------------------------------------------
* numele fisierului text include data, ora si minutul
* la care are loc salvarea
COPY PROCEDURES TO 'copie_ps_vinzari_'+DTOS(date())+'_'+;
    PADL(ALLT(STR(HOUR(DATETIME()))),2,'0')+;
    PADL(ALLT(STR(MINUTE(DATETIME()))),2,'0')+'.txt'
QUIT
*-----------------------------------------------------------------------
```

Lucrul cu variabile de memorie este grozav de indicat, deoarece căutarea şi sortarea este foarte rapidă. Să parcurgem pe rând declanşatoarele generalizate. Începem cu cel de ştergere, al cărui cod-sursă este prezentat în listingul 6.25.

Listing 6.25. Declanşatorul de ştergere valabil pentru toate tabelele bazei

```
*-----------------------------------------------------------------------
*       triggeru' general de stergere
*-----------------------------------------------------------------------
PROCEDURE trg_del
parameter tabela__
LOCAL i, copil_, tagcopil_, tabela_, curent_, tagparinte_, nrindexparinte_, ;
  cheieparinte_, valoare_cheieparinte_, sir_valoare_cheieparinte_

tabela_ = allt(upper(tabela__))
IF TYPE('vTabele') = "U" OR TYPE('vRelatii') = "U"
  do setup
ENDIF
trg_del_&tabela_ = .t.

set_deleted_ = SET('DELETE')
SET DELETE ON

*** cautam daca tabela curenta este una parinte
curent_ = 1
i = ASCAN(vParinti, tabela_, curent_)

DO WHILE i > 0
  * s-a gasit o noua tabela-copil pentru acest parinte
  tagparinte_ = ALLT(vTaguriParinte(i))
  nrindexparinte_ = TAGNO(tagparinte_, vParinti(i), vParinti(i))
  cheieparinte_ = KEY(nrindexparinte_,vParinti(i))
  valoare_cheieparinte_ = EVALUATE(vParinti(i)+"."+cheieparinte_)

  copil_ = ALLT(vCopii(i))
  tagcopil_ = ALLT(vTaguriCopil(i))

  IF UPPER(vParinti(i))='FACTURI' AND UPPER(copil_)='LINIIFACT' ;
      AND _TRIGGERLEVEL > 1
    * stergerea are loc in tabela FACTURI ca urmare a stergerii
    * ultimei linii (pt. aceasta factura) din tabela LINIIFACT

  ELSE
    * se cauta in indexul (tag-ul) copil daca exista o integistrare
    * care sa corespunda inegistrarii curente din parinte
    IF INDEXSEEK(valoare_cheieparinte_, .F., copil_, tagcopil_)
      DO CASE
      CASE TYPE(cheieparinte_) = 'C'
          sir_valoare_cheieparinte_ = valoare_cheieparinte_
      CASE TYPE(cheieparinte_) = 'N'
          sir_valoare_cheieparinte_ = ALLT(STR(valoare_cheieparinte_))
      ENDCASE
      MESSAGEBOX('Nu puteti sterge aceasta linia din '+vParinti(i)+chr(13)+ ;
          ' in care cheia de indexare este '+sir_valoare_cheieparinte_+chr(13)+ ;
          'deoarece exista linii copil in '+vCopii(i))
      SET DELETE &set_deleted_
      RETURN .F.
    ENDIF
  ENDIF
  curent_ = i + 1
  i = ASCAN(vParinti, tabela_, curent_)
ENDDO
```

```
** numai pentru LINIIFACT
IF tabela_ = 'LINIIFACT'
  LOCAL cite_, nrfact_, linie_
  DIME cite_(1,1), linie_(1,1)
  cite_ = 0
  linie_ = 0
  nrfact_ = liniifact.nrfact

  * se interzice stergerea altei linii decit ultima
  SELECT MAX(Linie) ;
  INTO ARRAY linie_ ;
  FROM LINIIFACT ;
  WHERE NrFact = nrfact_ AND !DELETED()

  SELECT liniifact
  IF linie_(1,1) # liniifact.linie
      MESSAGEBOX('Nu puteti sterge decit ultima linie din factura !')
      SET DELETE &set_deleted_
      RETURN .F.
  ENDIF

  * se verifica daca mai ramine vreo linie in factura
  SELECT COUNT(*) ;
  INTO ARRAY cite_ ;
  FROM LINIIFACT ;
  WHERE NrFact = nrfact_ AND !DELETED()

  IF cite_(1,1) = 1
      MESSAGEBOX('Aceasta a fost ultima linie din factura !')
      IF SEEK(nrfact_, 'facturi', 'nrfact')
          SELECT facturi
          REPLACE ValTotala WITH 0
          DELETE
      ENDIF
  ELSE
      * se actualizeaza FACTURI.ValTotala
      UPDATE facturi SET valtotala = valtotala - liniifact.valcutva ;
          WHERE nrfact = liniifact.nrfact
  ENDIF
  SELECT liniifact
ENDIF
trg_del_&tabela_ = .f.
SET DELETE &set_deleted_
RETURN .T.
ENDPROC
*-----------------------------------------------
```

La început se verifică dacă variabilele ce conţin informaţii despre relaţiile dintre tabele au fost iniţializate, scop pentru care se foloseşte funcţia `TYPE()`. La primul apel al vreunui declanşator se execută procedura `setup` prezentată anterior.

În continuare, se scanează vectorul `vParinti`, identificându-se toate restricţiile referenţiale în care tabela din care se şterge înregistrarea este părinte. Pentru fiecare situaţie, se caută dacă în tabela-copil există măcar o linie ce prezintă valoarea cheii străine identică cu cea a înregistrării de şters, caz în care se afişează un mesaj lămuritor şi se returnează valoarea `FALSE (.F.)`, ştergerea fiind deci interzisă. Căutarea este rapidă, deoarece se foloseşte funcţia `INDEXSEEK()` fără a se modifica pointerul tabelei-copil.

După secvenţa generală se introduce partea specifică declanşatorului de ştergere al tabelei LINIIFACT, precedată de `IF`-ul prin care secvenţa respectivă este executată numai în cazul acestei tabele.

Listingul 6.26 conţine corpul declanşatorului de inserare, în care se verifică dacă tabela în care se adaugă noua înregistrare apare drept copil în restricţii referenţiale. În caz că da, se verifică dacă există o linie-părinte – se foloseşte din nou funcţia `INDEXSEEK()` sau se testează mai întâi dacă valoarea cheii străine este egală cu valoarea de pe linia curentă a tabelei-părinte. Ultima situaţie este destul de obişnuită pentru, de exemplu, tabelele FACTURI şi LINIIFACT, în care după adăugarea unei înregistrări în părinte urmează adăugarea înregistrărilor-copil şi, în funcţie de modul de adăugare (`APPEND BLANK` sau `INSERT INTO...`) şi de tipul buffering-ului ales, înregistrarea din părinte nu a fost „comisă" (scrisă efectiv din buffer).

Listing 6.26. Declanşatorul general de inserare

```
*-----------------------------------------------
* declansator general pentru inserare
*-----------------------------------------------
PROCEDURE trg_ins
parameter tabela__
LOCAL i, copil_, tagcopil_, nrindexcopil_, cheiecopil_, valoare_cheiecopil_,;
  tabela_, curent_, parinte_, tagparinte_, nrindexparinte_, ;
  cheieparinte_, valoare_cheieparinte_, sir_valoare_cheiecopil_

tabela_ = allt(upper(tabela__))
IF TYPE('vTabele') = "U" OR TYPE('vRelatii') = "U"
  do setup
ENDIF
trg_ins_&tabela_ = .t.

set_deleted_ = SET('DELETE')
SET DELETE ON

*** cautam daca tabela curenta este una copil
local curent_
curent_ = 1
i = ASCAN(vCopii, tabela_, curent_)

DO WHILE i > 0
  * s-a gasit o (noua) tabela-parinte pentru acest copil
  copil_ = ALLT(vCopii(i))
  tagcopil_ = ALLT(vTaguriCopil(i))
  nrindexcopil_ = TAGNO(tagcopil_, vCopii(i), vCopii(i))
  cheiecopil_ = KEY(nrindexcopil_,vCopii(i))
  valoare_cheiecopil_ = EVALUATE(copil_+"."+cheiecopil_)

  parinte_ = ALLT(vParinti(i))
  tagparinte_ = ALLT(vTaguriParinte(i))
  nrindexparinte_ = TAGNO(tagparinte_, vParinti(i), vParinti(i))
  cheieparinte_ = KEY(nrindexparinte_,vParinti(i))
  valoare_cheieparinte_ = EVALUATE(parinte_+"."+cheieparinte_)

  DO CASE
  CASE TYPE(cheiecopil_) = 'C'
      sir_valoare_cheiecopil_ = valoare_cheiecopil_
```

```
   CASE TYPE(cheiecopil  ) = 'N'
        sir_valoare_cheiecopil_ = ALLT(STR(valoare_cheiecopil_))
   ENDCASE

   * se cauta in indexul (tag-ul) parinte daca exista o integistrare
   * care sa corespunda inegistrarii curente din parinte
   IF valoare_cheiecopil_ # valoare_cheieparinte_ AND ;
            !INDEXSEEK(valoare_cheiecopil_, .F., parinte_, tagparinte_)
        MESSAGEBOX('Noua valoare a '+cheiecopil_+' -'+sir_valoare_cheiecopil_+;
            '- din tabela '+vCopii(i)+chr(13)+ ;
            ' nu se regaseste in tabela parinte '+vParinti(i))
        SET DELETE &set_deleted_
        RETURN .F.
   ENDIF
   curent_ = i + 1
   i = ASCAN(vCopii, tabela_, curent_)
ENDDO

*** in tabela FACTURI se interzice o valoare diferita de zero a atributului ValTotala
IF tabela_ = 'FACTURI'
   IF facturi.valtotala # 0
        MESSAGEBOX('O factura noua nu poate avea decit valoarea zero'+;
            CHR(13)+'(este calculata automat ulterior)')
        SET DELETE &set_deleted_
        RETURN .F.
   ENDIF
ENDIF

IF tabela_ = 'LINIIFACT'

   * se verifica daca linia inserata respecta secventa
   LOCAL nrfact_, v1_
   DIME v1_(1,1)
   v1_ = 0
   nrfact_ = liniifact.nrfact
   SELECT MAX(linie) FROM LINIIFACT INTO ARRAY v1_ WHERE nrfact = nrfact_ ;
AND !DELETED()
   IF liniifact.linie > v1_(1,1) + 1
        MESSAGEBOX('Numarul de linie din factura nu respecta ordinea !')
        SET DELETE &set_deleted_
        RETURN .F.
   ENDIF

   * se actualizeaza FACTURI.ValTotala
   UPDATE facturi SET valtotala = valtotala + liniifact.valcutva ;
            WHERE nrfact = liniifact.nrfact
ENDIF

trg_ins_&tabela_ = .f.
SET DELETE &set_deleted_
RETURN .T.
ENDPROC
*-----------------------------------------------------------------------------
```

După cum observăm în listingul de mai sus, după secvența generală apar secvențele specifice tabelelor FACTURI și LINIIFACT prin care se interzice modificarea directă a atributului `FACTURI.ValTotala`, respectiv se incrementează `FACTURI.ValTotală` cu valoarea `LINIIFACT.ValCuTVA` de pe noua linie inserată.

În fine, ultimul dintre cele trei declanșatoare, cel de actualizare, este și cel mai dificil, deoarece, pe de o parte, trebuie să trateze ambele eventuale spețe ale tabelei – cea de copil și cea de părinte – în restricțiile referențiale în care este implicată și, în al doilea rând, pentru tabelele-părinte, să propage în cascadă modificările cheilor primare în toate tabelele-copil corespondente – vezi listingul 6.27.

Listing 6.27. Declanșatorul general de modificare

```
*-----------------------------------------------------------------------------
* declansator general ptr. modificare
*-----------------------------------------------------------------------------
PROCEDURE trg_upd
parameter tabela__
LOCAL i, copil_, tagcopil_, nrindexcopil_, cheiecopil_, valoare_veche_cheiecopil_,;
  valoare_noua_cheiecopil_, tabela_, curent_, parinte_, tagparinte_, nrindexparinte_, ;
  cheieparinte_, valoare_cheieparinte_, valoare_veche_cheieparinte_, ;
  valoare_noua_cheieparinte_, sir_valoare_cheiecopil_

tabela_ = allt(upper(tabela__))
IF TYPE('vTabele') = "U" OR TYPE('vRelatii') = "U"
  do setup
ENDIF
trg_upd_&tabela_ = .t.

set_deleted_ = SET('DELETE')
SET DELETE ON

*** cautam daca tabela curenta este una copil
local curent_
curent_ = 1
i = ASCAN(vCopii, tabela_, curent_)

DO WHILE i > 0

  * s-a gasit o (noua) tabela-parinte pentru acest copil
  copil_ = ALLT(vCopii(i))
  tagcopil_ = ALLT(vTaguriCopil(i))
  nrindexcopil_ = TAGNO(tagcopil_, vCopii(i), vCopii(i))
  cheiecopil_ = KEY(nrindexcopil_,vCopii(i))
  valoare_veche_cheiecopil_ = OLDVAL(cheiecopil_, copil_)
  valoare_noua_cheiecopil_ = EVALUATE(copil_+"."+cheiecopil_)

  IF valoare_veche_cheiecopil_ # valoare_noua_cheiecopil_ AND _TRIGGERLEVEL <= 1
      parinte_ = ALLT(vParinti(i))
      tagparinte_ = ALLT(vTaguriParinte(i))
      nrindexparinte_ = TAGNO(tagparinte_, vParinti(i), vParinti(i))
      cheieparinte_ = KEY(nrindexparinte_,vParinti(i))
      valoare_cheieparinte_ = EVALUATE(parinte_+"."+cheieparinte_)

      DO CASE
      CASE TYPE(cheiecopil_) = 'C'
          sir_valoare_cheiecopil_ = valoare_noua_cheiecopil_
      CASE TYPE(cheiecopil_) = 'N'
          sir_valoare_cheiecopil_ = ALLT(STR(valoare_noua_cheiecopil_))
      ENDCASE
```

```
        * se cauta in indexul (tag-ul) parinte daca exista o integistrare
        * care sa corespunda inegistrarii curente din parinte
        IF valoare_noua_cheiecopil_ # valoare_cheieparinte_ AND ;
            !INDEXSEEK(valoare_noua_cheiecopil_, .F., parinte_, tagparinte_)
            MESSAGEBOX('Noua valoare a '+cheiecopil_+' -'+sir_valoare_cheiecopil_+;
                    '- din tabela '+vCopii(i)+chr(13)+ ;
                    ' nu se regaseste in tabela parinte '+vParinti(i))
            SET DELETE &set_deleted_
            RETURN .F.
        ENDIF
    ENDIF
    curent_ = i + 1
    i = ASCAN(vCopii, tabela_, curent_)
ENDDO


*** daca tabela curenta este una parinte si se modifica cheia primara
** trebuie propagata modificarea in toti copiii
curent_ = 1
i = ASCAN(vParinti, tabela_, curent_)

DO WHILE i > 0

    * s-a gasit o (noua) tabela copil pentru acest parinte
    parinte_ = ALLT(vParinti(i))
    tagparinte_ = ALLT(vTaguriParinte(i))
    nrindexparinte_ = TAGNO(tagparinte_, vParinti(i), vParinti(i))
    cheieparinte_ = KEY(nrindexparinte_,vParinti(i))
    valoare_veche_cheieparinte_ = OLDVAL(cheieparinte_, parinte_)
    valoare_noua_cheieparinte_ = EVALUATE(parinte_+"."+cheieparinte_)

    * daca s-a modificat atributul-cheie, atunci actualizarea
    * se propaga in tabela copil
    IF valoare_veche_cheieparinte_ # valoare_noua_cheieparinte_
        copil_ = ALLT(vCopii(i))
        tagcopil_ = ALLT(vTaguriCopil(i))
        nrindexcopil_ = TAGNO(tagcopil_, vCopii(i), vCopii(i))
        cheiecopil_ = KEY(nrindexcopil_,vCopii(i))
        valoare_veche_cheiecopil_ = OLDVAL(cheiecopil_, copil_)
        valoare_noua_cheiecopil_ = EVALUATE(copil_+"."+cheiecopil_)

        UPDATE &copil_ ;
            SET &tagcopil_ = valoare_noua_cheieparinte_ ;
            WHERE &tagcopil_ = valoare_veche_cheiecopil_

    ENDIF
    curent_ = i + 1
    i = ASCAN(vParinti, tabela_, curent_)
ENDDO

* daca tabela e FACTURI
IF tabela_ = 'FACTURI'
    *** se interzice modificarea interactiva a atributului ValTotala
    valtotala_NEW = facturi.valtotala
    valtotala_OLD = OLDVAL('valtotala', 'facturi')
    IF valtotala_NEW # valtotala_OLD
        IF _TRIGGERLEVEL <= 1
            MESSAGEBOX('Nu puteti modifica interactiv valoarea totala a facturii !')
            SET DELETE &set_deleted_
```



```
            RETURN .F.
        ENDIF
    ENDIF
ENDIF

IF tabela_ = 'LINIIFACT'
    LOCAL nrfact_OLD, nrfact_NEW, codpr_OLD, codpr_NEW, ;
        valcutva_OLD, valcutva_NEW
    nrfact_OLD = OLDVAL('nrfact', 'liniifact')
    nrfact_NEW = liniifact.nrfact
    valcutva_OLD = OLDVAL('valcutva', 'liniifact')
    valcutva_NEW = liniifact.valcutva
    *** Daca s-a schimbat numarul facturii, linia trebuie scoasa de la fosta factura
    ** si adaugata la noua factura
    IF nrfact_NEW # nrfact_OLD  AND _TRIGGERLEVEL <= 1
        * se scade valoarea de la factura "veche"
        UPDATE facturi SET valtotala = valtotala - valcutva_OLD ;
            WHERE nrfact = nrfact_OLD
        * se aduna valoarea de la factura "noua"
        UPDATE facturi SET valtotala = valtotala + valcutva_NEW ;
            WHERE nrfact = nrfact_NEW
    ELSE  ** nu s-a modificat NrFact
        IF valcutva_OLD # valcutva_NEW
            UPDATE facturi SET valtotala = valtotala -  valcutva_OLD + valcutva_NEW ;
                    WHERE nrfact =     nrfact_OLD
        ENDIF
    ENDIF
ENDIF trg_upd_&tabela_ = .f.
SET DELETE &set_deleted_
ENDPROC
*-----------------------------------------------------------------------
```

În final sunt incluse elemente specifice modificării tabelelor FACTURI şi LINIIFACT. Toate procedurile stocate (listingul 6.28), în noua variantă, pot fi descărcate de la adresa http://www.es.uaic.ro/infoec/books.

Nu trebuie uitat ca şi numele declanşatoarelor să fie schimbat, procedura de refacere a bazei de date fiind modificată ca în listingul 6.29.

Listing 6.29. Re-declararea declanşatoarelor pentru tabelele bazei

```
*-----------------------------------------------------------------------
...

PUBLIC vTabele
DIME vTabele(1,1)
ADBOBJECTS(vTabele, "TABLE")
FOR i = 1 TO ALEN(vTabele)
    tab_ = vTabele(i)
    trigger_ = "trg_del_"+allt(tab_)+"()"
    CREATE TRIGGER ON &tab_ FOR DELETE AS &trigger_
    trigger_ = "trg_upd_"+allt(tab_)+"()"
    CREATE TRIGGER ON &tab_ FOR UPDATE AS &trigger_
    trigger_ = "trg_ins_"+allt(tab_)+"()"
    CREATE TRIGGER ON &tab_ FOR INSERT AS &trigger_
ENDFOR
...
```

Şi acest listing (integral) este disponibil la aceeaşi adresă web.

6.8. Optimizarea procedurilor stocate pentru aplicații ce rulează pe un singur calculator (locale)

La prezentarea funcțiilor stocate ce calculează valorile implicite ale unor atribute trebuie precizat că acestea pot afecta serios viteza aplicației. Probabil că, din acest punct de vedere, „ucigașul" numărul 1 este procedura `def_codpr_liniifact` din listingul 6.7. La dimensiuni mari ale tabelei LINIIFACT, este aproape sigur că această procedură va capota, dat fiind că la fiecare inserare în tabelă se calculează (prin grupare) cel mai frecvent produs, ceea ce e de-a dreptul hilar. Dacă procedura n-ar fi fost scrisă de noi, am fi avut cuvinte și mai grele...

Majoritatea situațiilor în care valorile implicite sau regulile de validare complexe (ce reclamă interogări costisitoare) sunt absolut necesare pot fi rezolvate prin optimizări care presupun, în general, folosirea variabilelor de memorie sau prin denormalizarea bazei de date.

Atunci când aplicația este monoutilizator, se poate insista pe folosirea variabilelor de memorie. De exemplu, la deschiderea aplicației se pot executa o serie de interogări prin care, în variabile publice, se stochează ultimul cod de client, ultimul număr de factură etc. Mai mult, se pot modifica funcțiile ce returnează valorile implicite ale atributelor respective. În listingul 6.30 este „expusă" a doua variantă a procedurii stocate ce întoarce codul unui nou client.

Listing 6.30. Varianta optimizată (*stand-alone*) a `def_codcl_clienti`

```
**************************************
PROCEDURE def_codcl_clienti
**************************************
LOCAL v1
DIME v1(1,1)
v1 = 0
IF TYPE('codcl_max_') = 'U'
  SELECT MAX(CodCl) INTO ARRAY v1 FROM clienti
  PUBLIC codcl_max_
  IF v1(1,1) = 0
      codcl_max_ = 1000
  ELSE
      codcl_max_ = v1(1,1)
  ENDIF
ENDIF
codcl_max_ = codcl_max_ + 1
RETURN codcl_max_
ENDPROC
```

Marele avantaj al acestei noi versiuni este că fraza `SELECT` ce parcurge întreaga tabelă CLIENTI se execută o singură dată pe sesiune, indiferent de câte înregistrări sunt adăugate.

Analog se modifică în tabela FACTURI procedurile stocate pentru calculul valorilor implicite ale atributelor `NrFact`, `CodCl` și `Gestiune`, ca în listingul 6.31. De remarcat că ultima funcție-valoare implicită trebuie declarată printr-o comandă `ALTER TABLE`:

```
ALTER TABLE facturi ALTER COLUMN gestiune SET DEFAULT
   def_gestiune_facturi()
```


Listing 6.31. Funcțiile stocate ce calculează valorile implicite pentru atributele `NrFact`, `CodCl` și `Gestiune` în tabela `FACTURI`

```
**************************************
PROCEDURE def_nrfact_facturi
**************************************
LOCAL v1
DIME v1(1,1)
v1 = 0
IF TYPE('nrfact_max_') = 'U'
  SELECT MAX(NrFact) INTO ARRAY v1 FROM facturi
  PUBLIC nrfact_max_
  IF v1(1,1) = 0
      nrfact_max_ = 1110
  ELSE
      nrfact_max_ = v1(1,1)
  ENDIF
ENDIF
nrfact_max_ = nrfact_max_ + 1
RETURN nrfact_max_
ENDPROC

*************************************************************************
PROCEDURE def_codcl_facturi
**************************************
LOCAL v1
DIME v1(1,1)
v1 = 0
IF TYPE('cel_mai_codcl_') = 'U'

  SELECT TOP 1 CodCl, COUNT(*) AS nr ;
  FROM FACTURI INTO ARRAY v1 ;
  GROUP BY CodCl ORDER BY nr DESC
  PUBLIC cel_mai_codcl_

  IF v1(1,1) = 0
      IF !USED('clienti')
          USE clienti IN 0
      ENDIF
      SELECT clienti
      IF RECCOUNT() = 0
          MESSAGEBOX('Nu puteti introduce o factura cita vreme nu aveti nici un client !')
          RETURN .F.
      ELSE
          IF EOF() OR BOF()
                GO TOP
          ENDIF
      ENDIF
      SELECT facturi
      cel_mai_codcl_ = clienti.codcl
  ELSE
      cel_mai_codcl_ = v1(1,1)
  ENDIF

ENDIF

RETURN cel_mai_codcl_
ENDPROC
```

```
*****************************************
PROCEDURE def_gestiune_facturi
*****************************************
LOCAL v1
DIME v1(1,1)
v1 = ' '
IF TYPE('cea_mai_gestiune_') = 'U'
  SELECT TOP 1 Gestiune, COUNT(*) AS nr ;
  FROM FACTURI INTO ARRAY v1 ;
  GROUP BY Gestiune ORDER BY nr DESC
  PUBLIC cea_mai_gestiune_

  IF v1(1,1) = ' '
      IF !USED('gestiuni')
          USE gestiuni IN 0
      ENDIF
      SELECT gestiuni
      IF RECCOUNT() = 0
          MESSAGEBOX('Nu puteti introduce o factura cita vreme nu e introdusa ;
nici o gestiune !')
          RETURN .F.
      ELSE
          IF EOF() OR BOF()
                GO TOP
          ENDIF
      ENDIF
      SELECT facturi
      cea_mai_gestiune_ = gestiuni.gestiune
  ELSE
      cea_mai_gestiune_ = v1(1,1)
  ENDIF
ENDIF

RETURN cea_mai_gestiune_
ENDPROC
*****************************************
```

Câştigul de productivitate este foarte important la dimensiuni mari ale tabelei. Însă punctul nevralgic al bazei rămâne tabela LINIIFACT, mai precis, regulile de validare ale acesteia, care blochează efectiv aplicaţia atunci când numărul înregistrărilor este de ordinul milioanelor. Funcţia de calcul al valorii implicite a atributului `NrFact` poate fi modificată după calapodul de mai sus – vezi listingul 6.32.

Listing 6.32. Funcţia stocată (modificată) ce calculează valorile implicite ale `LINIIFACT.NrFact`

```
*****************************************
PROCEDURE def_nrfact_liniifact
*****************************************
LOCAL v1
DIME v1(1,1)
v1 = 0
IF TYPE('nrfact_max_') = 'U'
  * daca variabila publica nu a fost initializata - se
  * determina factura curenta sau factura cea mai recenta
  IF !USED('facturi') OR EOF('facturi') OR BOF('facturi')
      SELECT MAX(NrFact) INTO ARRAY v1 FROM facturi
```



```
      PUBLIC nrfact_max
      IF v1(1,1) = 0
          MESSAGEBOX('Nu exista nici o factura !')
          RETURN .F.
      ELSE
          nrfact_max_ = v1(1,1)
      ENDIF
  ELSE
      * FACTURI este deschisa. se preia valoarea curenta
      nrfact_max_ = facturi.nrfact
  ENDIF
ELSE
  * se verifica daca nu s-a trecut la o alta factura
  IF !USED('facturi')
      USE facturi IN 0
  ENDI
  IF facturi.nrfact # nrfact_max_
      nrfact_max_ = facturi.nrfact
  ENDIF
ENDIF
RETURN nrfact_max_
ENDPROC
```

În noua versiune a funcţiei care întoarce valoarea implicită a atributului `LINIIFACT.Linie` – listingul 6.33 –, variabila publică include în numele său şi numărul facturii. După o primă inserare a unei linii în tabela LINIIFACT, se iniţializează variabila publică cu numele `linie_max_`*`nrfactura`*. Pentru sesiunea respectivă, următoarele inserări de noi linii pentru această factură vor folosi numai această variabilă, pe care o vor incrementa de fiecare dată în declanşatorul de inserare.

Listing 6.33. Funcţia stocată (modificată) ce calculează valorile implicite ale `LINIIFACT.Linie`

```
*****************************************
PROCEDURE def_linie_liniifact
*****************************************
LOCAL v1_, nrfact_, cNrFact_
DIME v1_(1,1)
v1_ = 0
nrfact_ = liniifact.nrfact
cNrFact_ = ALLT(STR(nrfact_))
nume_ = 'linie_max_'+cNrFact_
IF TYPE((nume_)) = 'U'
  * variabila publica nu a fost initializata
  SELECT MAX(linie) INTO ARRAY v1_ FROM liniifact WHERE nrfact = nrfact_ AND !DELETED()
  PUBLIC (nume_)
  &nume_ = v1_(1,1) + 1
ENDIF
RETURN &nume_
ENDPROC
```

Se poate pune problema: ce se întâmplă atunci când o linie este ştearsă? Variabila trebuie decrementată prin declanşatorul de ştergere al tabelei LINIIFACT. Din păcate, la următoarea adăugare de înregistrare, dacă nu s-a lansat între timp comanda PACK, cheia primară a tabelei va fi violată. Putem stabili, ca regulă de lucru, că, o dată ştearsă, o linie

trebuie fie eliminată fizic (prin PACK), fie modificată şi apoi recuperată (anulat marcajul de ştergere) – lucrul posibil în cadrul formularelor.

Iată că am ajuns la cea mai problematică dintre funcţiile care întorc valori implicite – `def_codpr_liniifact()`. Am testat funcţia pentru un număr de peste cinci milioane de înregistrări în tabela LINIIFACT şi blocajul a fost garantat. Mai întâi, va trebui să renunţăm la luxul de a ordona produsele după frecvenţa apariţiei lor în facturi, mulţumindu-ne ca valoarea `CodPr` să nu violeze unicitatea combinaţiei `(NrFact, CodPr)`.

Ideea ar fi ca, în cadrul unei sesiuni, la primul apel al funcţiei `def_codpr_liniifact` să se iniţializeze un vector ce conţine maximum 20 de produse care ar putea să apară pe factura respectivă fără a încălca restricţia de unicitate. Orice adăugare ulterioară ar putea să „recolteze" o valoare din acest vector. Pentru o bună funcţionare, este necesar să modificăm şi declanşatoarele tabelei LINIIFACT.

Listingul 6.34 conţine noua formă a `def_codpr_liniifact`. La prima inserare, din sesiunea curentă, a unei linii într-o factură se preiau în vector maximum 20 de valori ale `CodPr` , coduri ce nu sunt încă prezente în factura respectivă. Când factura este una nouă, vectorul conţine tocmai codurile primelor 20 de produse. Dacă în factură au fost deja preluate o serie de înregistrări în sesiuni anterioare, atunci vectorul se completează cu produse care nu fac parte din factură.

Listing 6.34. Funcţia stocată (modificată) ce calculează valorile implicite ale `LINIIFACT.CodPr`

```
*****************************************
PROCEDURE def_codpr_liniifact
*****************************************
LOCAL v1_, nrfact_, cNrFact_, nume_, nr_, nume1_
DIME v1_(1)
v1_ = 0
nrfact_ = liniifact.nrfact
cNrFact_ = ALLT(STR(nrfact_))
nume_ = 'vProduse_'+cNrFact_
nume1_ = nume_+'(1)'
IF TYPE((nume_)) = 'U' OR LEN((nume_)) = 0 OR  TYPE((nume1_)) # 'N'
  * variabila publica nu a fost initializata
  SELECT TOP 20 CodPr INTO ARRAY v1_ FROM PRODUSE WHERE CodPr NOT IN ;
      (SELECT CodPr FROM LINIIFACT WHERE nrfact = nrfact_ AND !DELETED() ) ORDER BY
CodPr

  IF _TALLY = 0
      MESSAGEBOX('Nu se mai pot introduce produse pe aceasta factura !')
      RETURN .F.
  ENDIF

  nr_ = MIN(20, ALEN(v1_))
  PUBLIC ARRAY &nume_ (nr_)
  = ACOPY (v1_, (nume_), 1, nr_)

ENDIF
RETURN &nume_ (1)
ENDPROC
```

Mai rămân de modificat declanşatoarele tabelei LINIIFACT, astfel:

- în cel de inserare – se elimină din vector codul produsului tocmai inserat;

- în cel de ştergere – se adaugă în vector codul produsului şters;
- în cel de modificare – se şterge codul produsului nou şi se adaugă codul produsului vechi.

Toate aceste modificări sunt prezentate în listingul 6.35.

Listing 6.35. Modificări apărute în corpul declanşatoarelor

```
*-----------------------------------------------------------------
* declansator general pentru inserare - modificat
*-----------------------------------------------------------------
PROCEDURE trg_ins

...

IF tabela_ = 'LINIIFACT'

  * se verifica daca linia inserata respecta secventa
  LOCAL nrfact_, v1_
  DIME v1_(1,1)
  v1_ = 0
  nrfact_ = liniifact.nrfact
  SELECT MAX(linie) FROM LINIIFACT INTO ARRAY v1_ WHERE nrfact = nrfact_ ;
AND !DELETED()
  IF liniifact.linie > v1_(1,1) + 1
      MESSAGEBOX('Numarul de linie din factura nu respecta ordinea !')
      SET DELETE &set_deleted_
      RETURN .F.
  ENDIF

  * se actualizeaza FACTURI.ValTotala
  UPDATE facturi SET valtotala = valtotala + liniifact.valcutva ;
          WHERE nrfact = liniifact.nrfact

  * se elimina din vectorul vProduse_&nrfact codul produsului tocmai inserat
  LOCAL cNrFact_, nume_, nr_
  nrfact_ = liniifact.nrfact
  cNrFact_ = ALLT(STR(nrfact_))
  nume_ = 'vProduse_'+cNrFact_
  IF TYPE((nume_)) # 'U'
      * variabila publica a fost initializata
      ADEL((nume_), ASCAN ((nume_), liniifact.codpr))
      nr_ = ALEN ( (nume_), 0)
      DIME &nume_ ( MAX((nr_ - 1),1))
  ENDIF

  * se incrementeaza variabila dedicată liniei
  nume_ = 'linie_max_'+cNrFact_
  IF TYPE((nume_)) # 'U'
      &nume_ = &nume_ + 1
  ENDIF
ENDIF

trg_ins_&tabela_ = .f.
SET DELETE &set_deleted_
RETURN .T.
*-----------------------------------------------------------------
```

```
*-------------------------------------------------------------------------------
* declansatorul (modificat) ptr. modificarea tabelelor
*-------------------------------------------------------------------------------
PROCEDURE trg_upd

...

IF tabela_ = 'LINIIFACT'

  ....

  * nu se permite modificarea liniei !
  IF liniifact.linie # OLDVAL('linie', 'liniifact')
      MESSAGEBOX('Numarul de linie din factura nu se poate modifica !')
      SET DELETE &set_deleted_
      RETURN .F.
  ENDIF

  * se elimina din vectorul vProduse_&nrfact codul produsului nou
  LOCAL cNrFact_, nume_, nr_, nume1_
  nrfact_ = liniifact.nrfact
  cNrFact_ = ALLT(STR(nrfact_))
  nume_ = 'vProduse_'+cNrFact_
  IF TYPE((nume_)) # 'U'
      * variabila publica a fost initializata
      ADEL((nume_), ASCAN ((nume_), liniifact.codpr))
      nr_ = ALEN ( (nume_), 0)
      DIME &nume_ ( MAX((nr_ - 1),1))
  ENDIF
  * se adauga in vectorul vProduse_&nrfact codul produsului vechi
  nume1_ = nume_+'(1)'
  IF TYPE((nume_)) # 'U'
      * variabila publica a fost initializata
      nr_ = ALEN ( (nume_), 0)
      IF nr_ = 1 AND TYPE((nume1_)) # "N"
          nr_ = 0
      ENDIF
      DIME &nume_ ( nr_ + 1 ) )
      &nume_ ( nr_ + 1 ) = OLDVAL('codpr', 'liniifact')
  ENDIF
ENDIF
trg_upd_&tabela_ = .f.
SET DELETE &set_deleted_
ENDPROC
*-------------------------------------------------------------------------------


*-------------------------------------------------------------------------------
* declansatorul (modificat) ptr. stergere
*-------------------------------------------------------------------------------
PROCEDURE trg_del

...

** numai pentru LINIIFACT
IF tabela_ = 'LINIIFACT'
```



```
...

  * se decrementeaza variabila cu numarul de linii pentru aceasta factura
  LOCAL v1_, nrfact_, cNrFact_, nume_
  nrfact_ = liniifact.nrfact
  cNrFact_ = ALLT(STR(nrfact_))
  nume_ = 'linie_max_'+cNrFact_
  IF TYPE((nume_)) # 'U'
      &nume_ = &nume_ - 1
  ENDIF

  * se re-adauga in vectorul vProduse_&nrfact codul produsului tocmai sters
  LOCAL nr_, nume1_
  nume_ = 'vProduse_'+cNrFact_
  nume1_ = nume_+'(1)'
  IF TYPE((nume_)) # 'U'
      * variabila publica a fost initializata
      nr_ = ALEN ( (nume_), 0)
      IF nr_ = 1 AND TYPE((nume1_)) # "N"
          nr_ = 0
      ENDIF
      DIME &nume_ ( nr_ + 1 ) )
      &nume_ ( nr_ + 1 ) = liniifact.codpr
  ENDIF

ENDIF
trg_del_&tabela_ = .f.
SET DELETE &set_deleted_
RETURN .T.
ENDPROC
*-------------------------------------------------------------------------------
```

Procedurile stocate în această ultimă versiune pe care am discutat-o pot fi descărcate de la adresa web http://es.uaic.ro/infoec/books.

Capitolul 7

Rapoarte în Visual FoxPro

În general, un raport este perceput ca fiind o listă sau o structură de afişare (sau format de listare) a unor informaţii (înregistrări) provenind din tabelele bazei de date.

7.1. Descrierea structurii unui raport

În Visual FoxPro, rapoartele (obiectele `Report` din `Project Manager`) pot fi percepute ca având o structură dual:

- pe de o parte, structura de date, care poate consta din una sau mai multe tabele din baza de date, între care există eventual relaţii permanente;
- pe de altă parte, structura de afişare (listare) care se referă la formatul efectiv al raportului sau, mai exact, la obiectele prin care se face legătura cu structura de date şi benzile (zonele) care compun raportul.

Clasic, *structura de date* a unui raport este specificată prin *mediul de date* (*data environment*) asociat fiecărui raport (vezi figura 7.1).

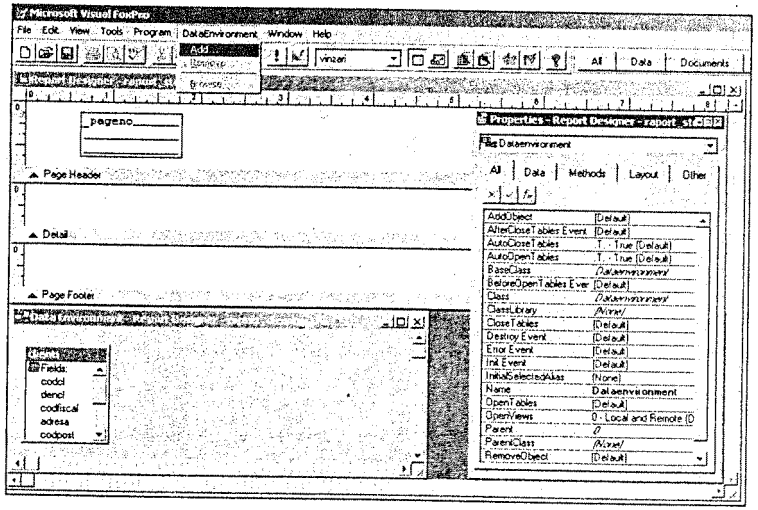


Figura 7.1. Structura de date a unui raport – `Data Environment`

În `Data Environment` pot fi aduse tabele sau *view*-uri din baza de date. Trebuie precizat faptul că atunci când se doreşte crearea unui raport mai complex, care implică date din mai multe tabele (deci, implicit, înregistrările din cele două tabele trebuie corelate), iar între cele două tabele este stabilită o relaţie permanentă, în `Data Environment` va fi moştenită şi respectiva relaţie.

7.1.1. Structura de date

Evident, acest demers presupune în primul rând crearea obiectului de tip raport căruia îi va fi asociată respectiva structură de date. În acest scop se poate apela fie la comanda `CREATE REPORT`, fie se poate invoca din meniul `File` opţiunea `New`, din fereastra care se va deschide selectându-se butonul radio `Report`, după care din dialogul care urmează se va alege butonul sub care este inscripţionat simplu `New Report` (mergându-se astfel pe o variantă „manuală"). Dacă se lucrează ordonat, adică integrând toate obiectele într-un proiect, se poate alege direct din cadrul de pagină `Documents` (sau `Docs`) nodul `Reports`, după care se acţionează butonul `New...`.

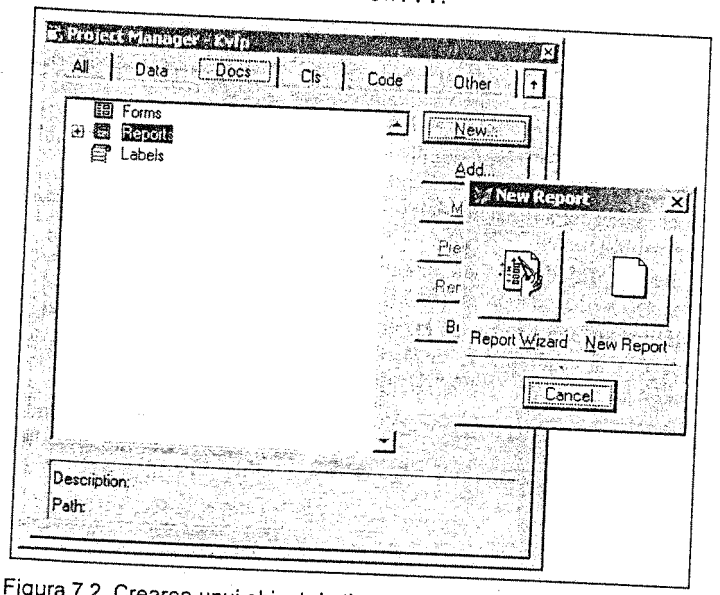


Figura 7.2. Crearea unui obiect de tip raport din `Project Manager`

În acest fel se va lansa de fapt instrumentul `Report Designer`, utilitarul cu care se pot crea rapoarte în Visual FoxPro.

Dar cum se creează în cele din urmă structura de date? Dacă din meniul principal al ferestrei Visual FoxPro se alege opţiunea `View→Data Environment`, sau din meniul contextual (click-dreapta) al generatorului de rapoarte se alege opţiunea `Data Environment`, se va deschide o fereastră specifică ce se va comporta ca un container pentru sursele de date ale raportului (tabele sau *view*-uri). Adăugarea unei tabele sau *view* din baza de date în structura de date a raportului se poate face prin opţiunea `Add...`, care se găseşte în submeniul `Data Environment` ce se activează în meniul principal VFP o dată cu deschiderea ferestrei `Data Environment`, sau prin opţiunea `Add...` din meniul contextual (click-dreapta) al ferestrei `Data Environment`. În acest fel se poate obţine o structură de date cum este cea din figura 7.3 (se observă aici şi importul legăturilor permanente dintre tabele).

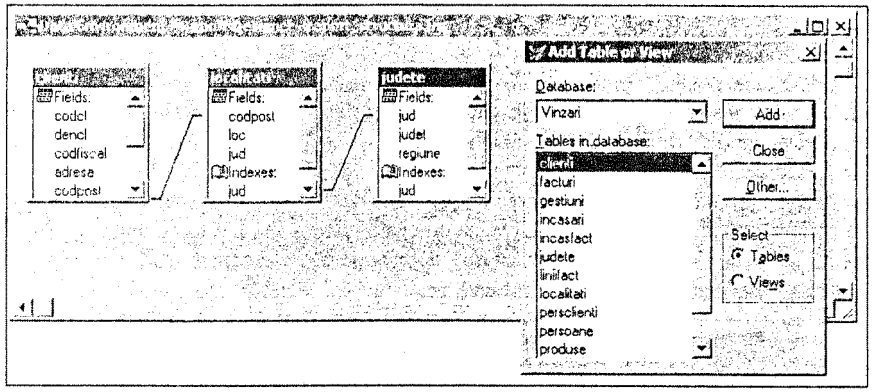


Figura 7.3. Adăugarea tabelelor în `Data Environment`

Iată câteva dintre principalele avantaje ale configurării structurii de date folosind `Data Environment`:

- adăugarea directă a tabelelor sau *view*-urilor din baza de date printre sursele de date ale raportului;
- gestiune simplificată a tabelelor care însoțesc raportul (deschiderea și închiderea lor automată);
- facilități avansate pentru utilizatori de a invoca cod în anumite evenimente sau metode ce însoțesc sursele de date sau containerul `Data Environment` (vezi figura 7.4).

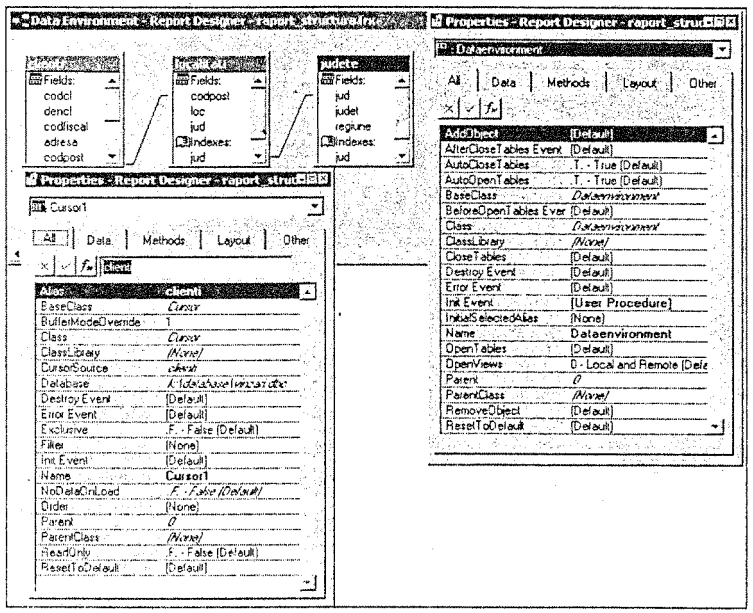


Figura 7.4. Proprietățile containerului `Data Environment` și ale unei surse de date

Utilizatorii familiarizați cu aplicații mai complexe și mai personalizate (ca să nu spunem „customizate") își vor pune inevitabil problema modului în care se poate preciza o structură de date care derivă, spre exemplu, dintr-o interogare (multe tabele, multe *join*-uri, multe filtre). Din acest punct de vedere, abordarea clasică descrisă mai sus are anumite limite. O soluție ar fi construirea unui *view* neactualizabil care să reflecte respectiva interogare. Dacă însă vom avea de-a face cu prea multe rapoarte de acest gen, vom ajunge la un moment dat să ne „încurcăm" în prea multe *view*-uri. Din experiența noastră, majoritatea dezvoltatorilor folosesc de regulă fraze SQL pentru a preciza sursa de date a unui raport. Unde și cum pot fi precizate asemenea fraze ? Iată două variante:

1. Construirea unei rutine simple (un program) în care să se precizeze: fraza `SELECT-SQL` plus comanda `REPORT FORM`, eventual precizând și clauza `PREVIEW`, care trimite raportul mai întâi pe ecran înainte de a-l lista efectiv. În fraza `SELECT` se va specifica prin clauza `INTO CURSOR` numele unei tabele temporare ce va găzdui datele înainte de a fi afișate prin raport (vezi listingul 7.1).

Listing 7.1. Rutina de listare a raportului `Rep_Clienti`

```
SELECT c.codcl, c.dencl, c.adresa, l.loc, j.judet, c.codpost, c.telefon;
from clienti c inner join localitati l inner join judete j;
on l.jud = j.jud on c.codpost = l.codpost;
INTO CURSOR c_clienti
REPORT FORM Rep_Clienti PREVIEW
```

2. Folosirea evenimentului `Init` al containerului `Data Environment` asociat raportului (vezi figura 7.5).

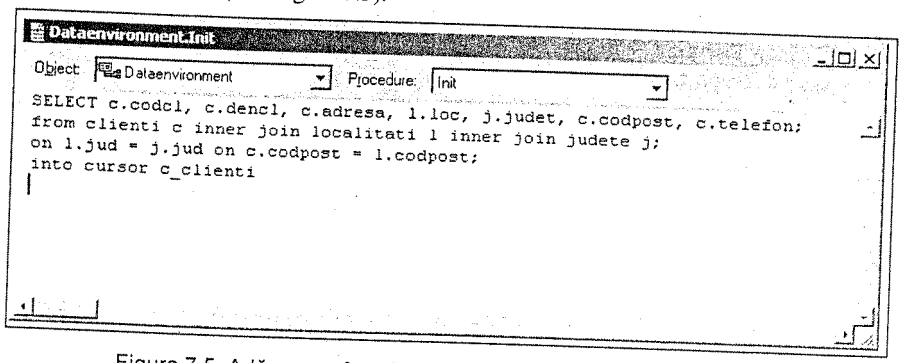


Figura 7.5. Adăugarea frazei `SELECT-SQL`, care va forma sursa de date în evenimentul `Init` al containerului `Data Environment`

În cazul în care nu se specifică în mod explicit (prin `Data Environment` sau metoda `Init`) nici o sursă de date, raportul va încerca să-și preia datele de care are nevoie din tabela (sau tabela derivată) deschisă în zona curentă de lucru (vezi capitolul 3 pentru eventuale lămuriri). Bineînțeles, dacă zona curentă de lucru nu furnizează structura de date pe care este configurat raportul, vor apărea mesaje de eroare la previzualizare sau listare.

7.1.2. Structura de afişare

Suprafaţa de afişare dintr-un raport este structurată de către `Report Designer` în mai multe zone care se diferenţiază între ele prin mai multe caracteristici, cum ar fi, de exemplu, **nivelul de repetitivitate**. Implicit, instrumentul `Report Designer` afişează rapoartele structurate în următoarele zone sau benzi:

- `Page Header` – în care vor fi afişate informaţii ce se vor regăsi în antetul fiecărei pagini;
- `Detail` – în care vor fi afişate informaţii ce se vor regăsi pe fiecare linie a raportului şi care formează secţiunea principală a raportului;
- `Page Footer` – în care vor fi afişate informaţii ce se vor regăsi în subsolul fiecărei pagini.

În funcţie de complexitatea raportului, mai pot fi activate încă patru benzi (sau zone):

- `Title` – în care se vor regăsi informaţii listate la începutul raportului o singură dată; de regulă, aici se afişează titlul raportului şi alte informaţii generale;
- `Summary` – în care se vor regăsi informaţii listate la sfârşitul raportului o singură dată; de regulă, aici sunt afişate informaţii sintetizate ce rezumă întregul raport (de exemplu, totalurile generale) sau concluzii;
- `Group Header` – în care se vor regăsi informaţii ce trebuie afişate la începutul fiecărui grup de înregistrări. De exemplu, clienţii pot fi grupaţi în funcţie de judeţul în care îşi au sediul, astfel că includerea informaţiilor despre judeţ în acestă secţiune va creşte claritatea raportului, acestea nefiind listate decât o singură dată la începerea grupului de clienţi al judeţului respectiv;
- `Group Footer` – în care se vor regăsi informaţii sintetizate la nivelul fiecărui grup din raport.

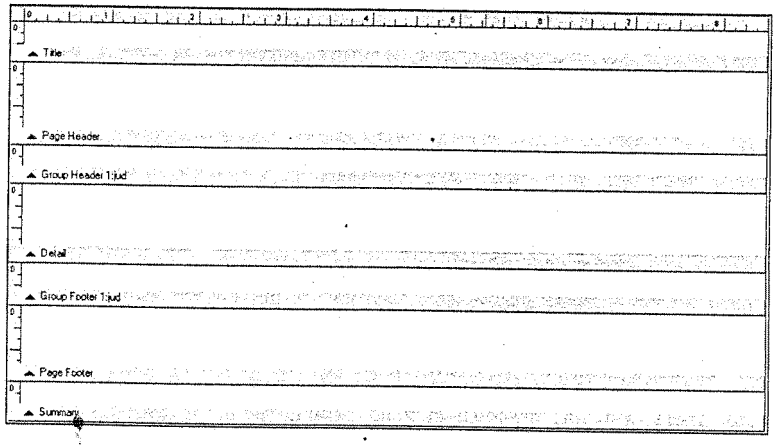


Figura 7.6. Secţiunile unui raport Visual FoxPro

Informaţiile dintr-un raport sunt precizate prin intermediul unor obiecte specifice. În acest scop se foloseşte bara de instrumente `Report Controls`, care de regulă se

activează o dată cu pornirea `Report Designer`-ului. Dacă această bară de instrumente nu este activată, atunci va trebui selectată opţiunea `Report Control Toolbar` din submeniul `View`. Controalele ce formează această bară de instrumente sunt prezentate în tabelul 7.1.

Tabelul 7.1. Structura barei de instrumente `Report Controls`

Controale	*Nume*	*Funcţie*
	Select Objects	Permite selectarea unuia sau mai multor obiecte din benzile raportului
	Label Control	Inserează blocuri de text
	Field Control	Inserează în raport câmpuri care conţin expresii ce pot face trimitere la: – câmpuri din tabele; – variabile sistem (gen `_pageno`); – variabile globale din aplicaţie; – funcţii predefinite gen `DATE()` etc. – funcţii definite de utilizator
	Line Control	Desenează o linie în raport
	Rectangle Control	Desenează un dreptunghi
	Rounded Rectangle Control	Desenează o elipsă
	Picture/OLE Bound Control	Inserează de regulă o imagine
	Button Lock	Inserează în raport mai multe controale de acelaşi tip fără să fie necesară reselectarea butonului din bara de instrumente Report Controls

Prin urmare, în raport sunt inserate trei categorii de obiecte:

- blocuri text;
- câmpuri care preiau de regulă informaţii din structura de date;
- elemente grafice: linii, dreptunghiuri, imagini.

Construirea unui raport cu aceste obiecte este ilustrată în figura 7.7.

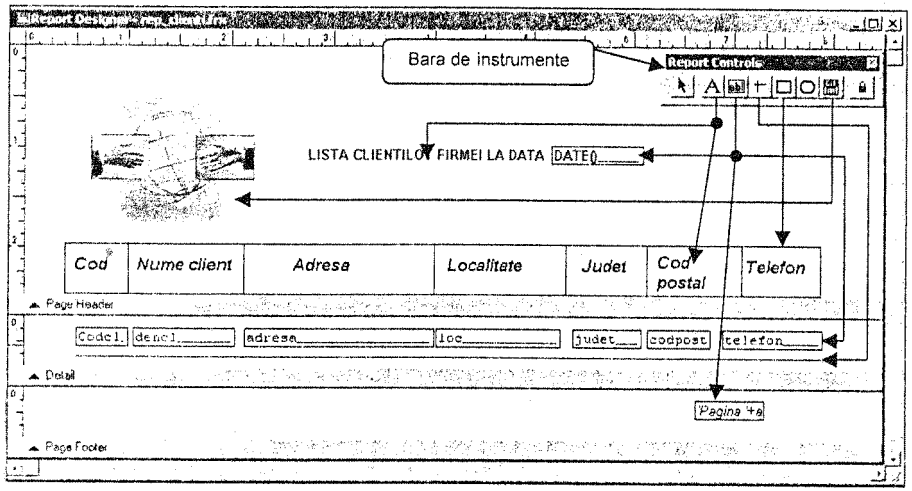


Figura 7.7. Raport construit folosind controalele din bara `Report Controls`

7.2. Definirea unui raport simplu

În acest paragraf ne propunem să parcurgem mai pe îndelete paşii ce trebuie urmaţi pentru construirea unui raport, pentru început, simplu. Ca punct de pornire vom considera că raportul va trebui să satisfacă următoarea cerinţă esenţială: obţinerea listei clienţilor firmei. Procesul pe care îl vom parcurge în cele ce urmează poate fi structurat în şase paşi. Astfel:

1. Este recomandabil ca mai întâi să fie desenată o schiţă din care să reiasă caracteristicile esenţiale ale raportului:
   - titlul raportului va fi: „Lista clienţilor firmei la data" curentă;
   - raportul va fi structurat pe următoarele cinci coloane: codul fiscal al clientului, numele acestuia, adresa completă, codul poştal şi telefonul;
   - în antetul raportului trebuie să apară şi sigla firmei;
   - dacă este structurat pe mai multe pagini, să fie indicat şi numărul paginii curente;
   - formatul paginii: A4.
2. Apoi se analizează structura de date a raportului, adică *de unde provin datele?*

În cazul nostru, sarcina este destul de simplă: toate datele provin din tabela `CLIENTI` şi anume din câmpurile: `codfiscal, dencl, adresa, codpost, telefon`.

3. În cele din urmă trebuie să trecem la construirea obiectului de tip *Report*.

Dacă obiectele aplicaţiei sunt organizate ordonat într-un proiect, atunci recomandabilă este selectarea nodului `Reports` din cadrul de pagină `Documents` al `Project Manager`-ului şi apăsarea butonului `New`. După care, din fereastra de dialog apărută se va selecta butonul `New Report`.

4. Pasul următor constă în specificarea surselor de date pentru raport.

În acest sens, după cum s-a arătat mai înainte, există cel puţin două variante. Prima ar fi deschiderea containerului `Data Environment` (opţiunea `View→Data Environment`) şi aducerea tabelei `CLIENTI` ca sursă de date (opţiunea `Data Environment→Add`). A doua constă în folosirea unei fraze `SELECT-SQL` pentru a obţine sursa de date.

5. În această etapă vom stabili structura de afişare.

Deocamdată nu ne vom complica „existenţa" şi vom proceda în felul următor:

- titlul şi sigla firmei le vom plasa în secţiunea `Page Header`, fiind conştienţi că aceste informaţii se vor repeta în antetul fiecărei pagini; tot aici vom include şi un cap de tabel prin care se va specifica numele fiecărei coloane din raport;
- coloanele raportului vor fi create prin plasarea unui obiect de tip `field` pentru fiecare câmp din tabela `CLIENTI` ce trebuie inclus în raport în secţiunea `Detail`;
- în secţiunea `Page Footer` va fi inclus un obiect de tip câmp de raport (`field`) care va conţine o expresie prin care să se obţină numărul paginii curente.

Mai întâi vom crea titlul raportului în felul următor: selectăm din bara de instrumente `Report Controls` butonul `Label` şi facem click cu mouse-ul în `Page Header`, în regiunea din care ar trebui să înceapă titlul. Va rezulta astfel un cursor de scriere şi vom folosi tastatura pentru a introduce textul „Lista clienţilor firmei", după cum se ilustrează în figura 7.8 (1). Dacă dorim anumite caracteristici specifice legate de fonturi, selectăm textul deja introdus şi apelăm la opţiunea `Format→Font...` – vezi figura 7.8 (2).

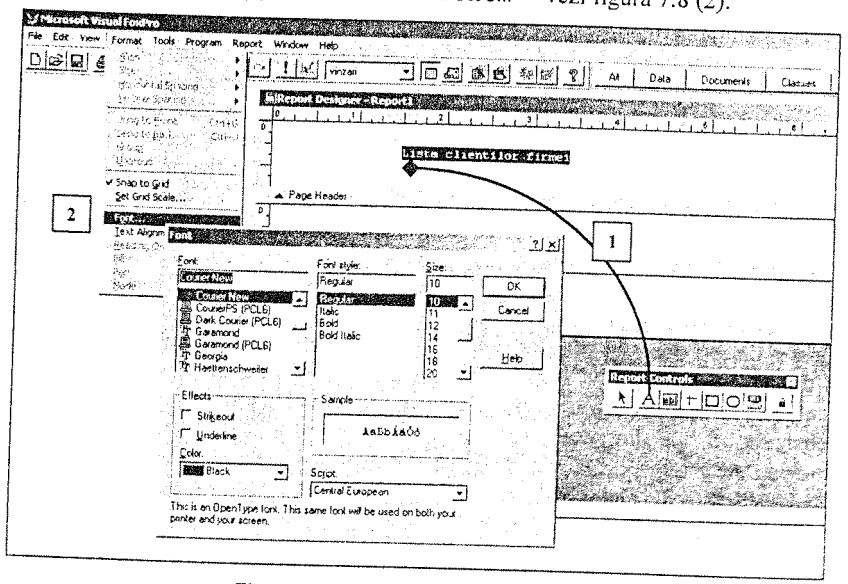


Figura 7.8. Crearea titlului raportului

Pentru a crea capul de tabel, vom introduce câte o etichetă pentru fiecare coloană în acelaşi mod în care am procedat pentru titlul raportului. Vom trasa apoi un chenar (butonul `Rectangle` din bara de instrumente a generatorului de rapoarte) – vezi figura 7.9. Dacă se doreşte un chenar cu marginile rotunjite, se foloseşte butonul `Rounded Rectangle` din bara de instrumente.

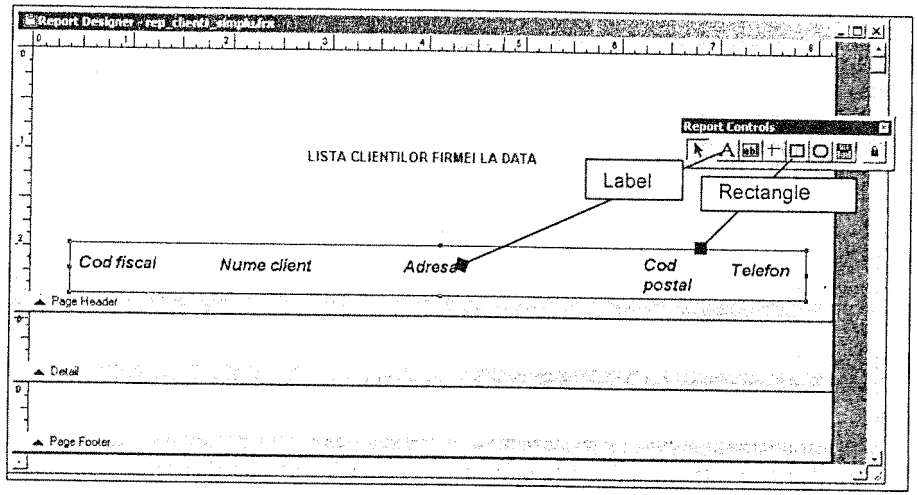


Figura 7.9. Crearea capului de tabel

Pentru reeditarea (modificarea) unei etichete se procedează în felul următor: mai întâi se va selecta butonul `Label` din `Report Controls`, după care se punctează, folosind cursorul mouse-ului, eticheta vizată, obținându-se astfel cursorul de scriere.

În continuare vom crea obiectele de tip *câmpuri de raport* necesare pentru definirea coloanelor raportului sau pentru introducerea datei curente şi numărului de pagină. Procedura pe care va trebui să o urmăm este următoarea (vezi şi figura 7.10): se selectează din bara de instrumente `Report Controls` butonul `Field` (1), după care se desenează un dreptunghi de lungimea coloanei ce urmează a fi create în regiunea destinată acesteia (2). Se va deschide astfel fereastra `Report Expression` (3), iar în căsuța de text în stânga căreia se găseşte eticheta `Expression` se va preciza o expresie ce va furniza informația care va fi afişată în respectiva coloană. Dacă din pasul patru (definirea structurii de date) s-a ales a doua variantă, atunci pur şi simplu se va introduce aici numele câmpului vizat din tabela-sursă. Însă, dacă în pasul patru s-a optat pentru prima soluție, se poate invoca instrumentul `Expression Builder` selectând butonul etichetat cu trei puncte de suspensie („. . .") din dreapta căsuței de text `Expression` (4). În fereastra `Expression Builder` există o listă cu câmpurile tabelei-sursă din `Data Environment`, de unde se poate selecta prin dublu-click câmpul dorit (5). Aceste operații sunt exemplificate în figura 7.10, şi se vor repeta pentru fiecare coloană a raportului. De asemenea, pentru data curentă se repetă aceeaşi procedură, cu deosebirea că în `Expression Builder` se va deschide lista combo-box-ului `Date` din regiunea `Functions`, unde se va căuta funcția `Date()`, iar pentru pagina curentă, tot în `Expression Builder`, din lista `Variables` se va selecta variabila de sistem `_pageno` (vezi figura 7.11).

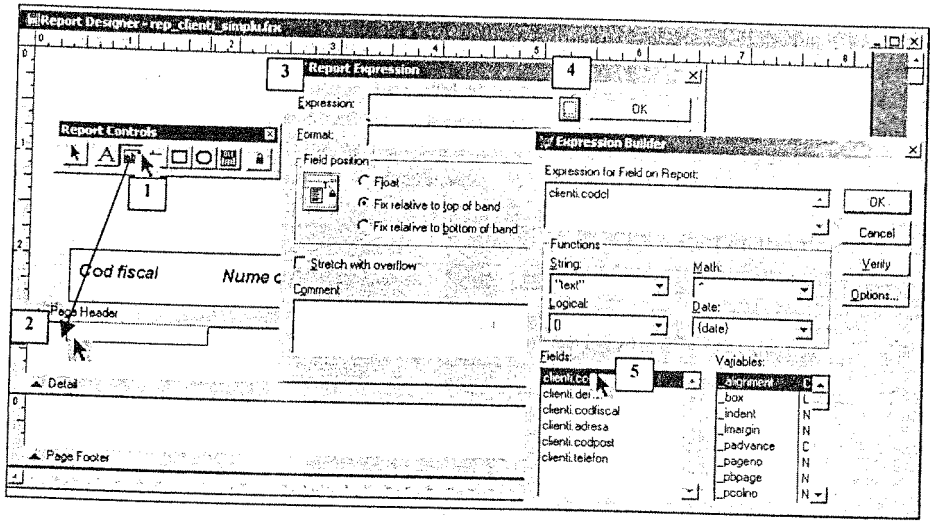


Figura 7.10. Adăugarea unui câmp în raport

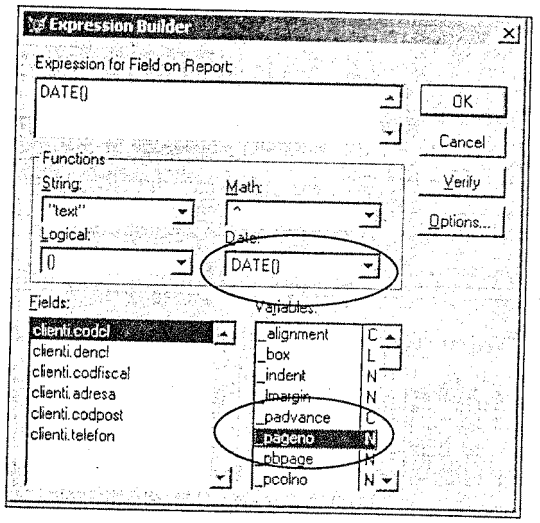


Figura 7.11. Localizarea funcției `Date()` şi a variabilei `_pageno`

Pentru a face mai inteligibilă informația, în câmpul din raport destinat numărului de pagină se poate preciza expresia `"pagina"+allt(str(_pageno))`.

În fine, pentru ca raportul să aibă un aspect cât de cât mai profesionist, se aliniază toate obiectele cu atenție, se adaugă o linie (butonul `Line` din bara de instrumente) sub câmpurile de raport corespunzătoare coloanelor raportului şi se include un obiect de tip OLE pentru a include şi sigla dorită. Acest obiect de tip OLE va face legătura cu un fişier de tip `.BMP`, iar procedura de urmat se derulează astfel (vezi şi figura 7.12): se selectează

mai întâi butonul Picture/OLE Bound Control din bara de instrumente a generatorului de rapoarte (1), se desenează apoi un dreptunghi corespunzător în zona destinată siglei (2). Ca urmare, se va deschide fereastra Report Picture (3), unde, în căsuța de text din cadrul Picture from, în stânga căreia se află butonul radio File, se precizează calea până la fișierul care conține sigla, sau se selectează butonul etichetat cu trei puncte de suspensie din dreapta (3), care deschide o fereastră Open. În această fereastră se poate naviga până la fișierul care conține imaginea dorită (5). Această manevră este exemplificată în figura 7.12.

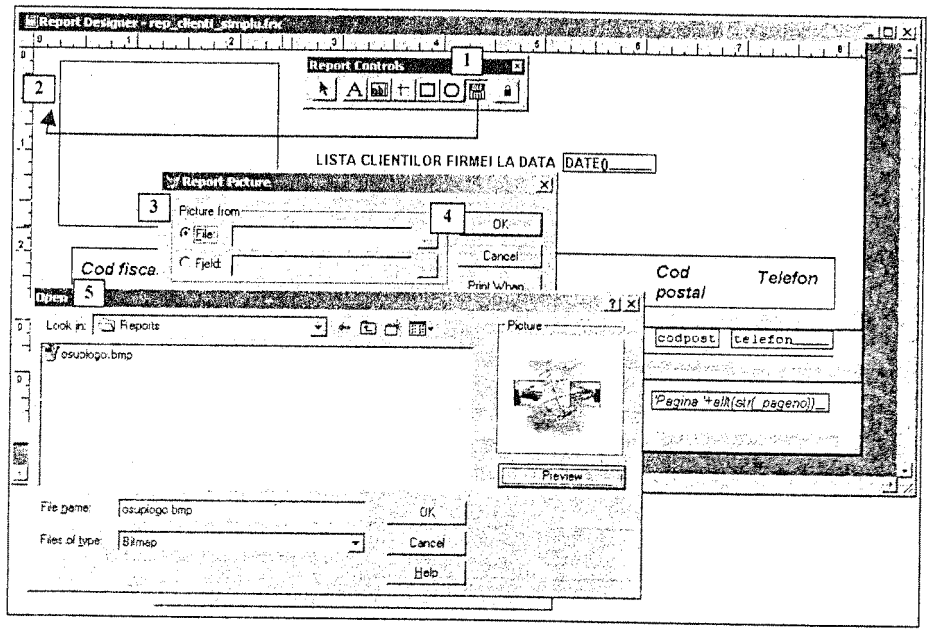


Figura 7.12. Adăugarea unei imagini într-un raport

6. În final, înainte de listarea propriu-zisă la imprimantă, vom previzualiza raportul pentru a verifica eventualele erori run-time și așezarea efectivă în pagină.

În acest ultim pas se mai pot aduce ajustări privitoare la grafica raportului și, de asemenea, se pot stabili câteva specificații privitoare la formatul de listare.

Specificațiile privitoare la formatul de listare se stabilesc prin opțiunea File→Page Setup... – vezi figura 7.13 (1). În fereastra Page Setup (2) se pot preciza:

- numărul de coloane de listare, cadrul Columns, spinnerul Number;
- lățimea unei coloane de listare, cadrul Columns, spinnerul Width;
- eventual, distanța dintre coloanele de listare, cadrul Columns, spinnerul Spacing;
- marginea din stânga a paginii, spinnerul Left Margin.

Pentru stabilirea formatului (dimensiunii) și a modului de orientare a paginii, din fereastra Page Setup se acționează butonul Print Setup... (3), care va deschide

fereastra Print Setup... (4) – bineînțeles, dacă aveți deja instalată o imprimantă (exemplificare în figura 7.13). Dintre opțiunile existente în acestă fereastră se pot evidenția:

- cadrul Paper, din care se poate stabili dimensiunea paginii (Letter, A4, B5, A3 etc.) din lista combinată în stânga căreia se găsește eticheta Size;
- cadrul Orientation cu cele două opțiuni Portrait (latura mică a hârtiei este latura de bază) și Landscape (latura mare a hârtiei este latura de bază).

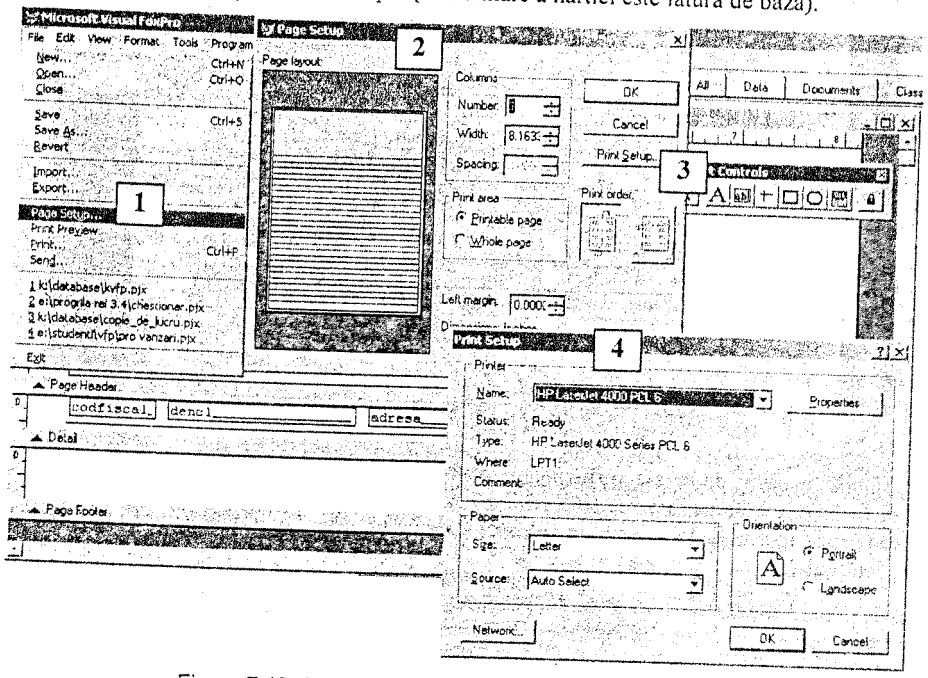


Figura 7.13. Stabilirea formatului de listare la imprimantă

În cele din urmă, pentru previzualizarea raportului (dacă în pasul patru s-a ales prima variantă), fie vom căuta opțiunea Preview din submeniul View sau opțiunea Print Preview din submeniul File, fie vom acționa butonul Preview... din Project Manager sau butonul Print Preview din bara de instrumente standard VFP. De regulă, după apelarea opțiunii de previzualizare se deschide o fereastră specifică însoțită de bara de instrumente Print Preview (vezi figura 7.14). Înainte de previzualizarea efectivă, sistemul va cere salvarea raportului. În urma acestei operațiuni vor rezulta două fișiere: unul cu extensia .frx și altul cu extensia .frt.

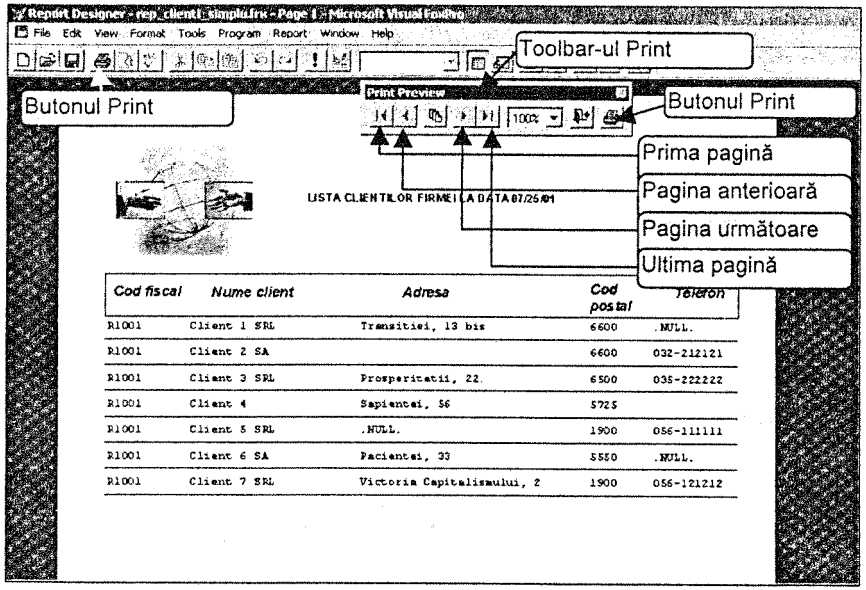


Figura 7.14. Fereastra de previzualizare

Dacă în pasul patru s-a optat pentru a doua variantă, formarea structurii de date va consta în deschiderea, înainte de previzualizare, a tabelei CLIENTI la care fac referire câmpurile raportului. În acest scop se poate scrie un program scurt de genul celui prezentat în listingul 7.2.

Listing 7.2. Programul de listare a raportului `Rep_Clienti_Simplu`

```
IF !USED('clienti')
    USE clienti IN 0
ENDIF
SELECT clienti
REPORT FORM rep_clienti_simplu PREVIEW
```

Sintaxa comenzii `REPORT FORM` este următoarea:

```
REPORT FORM nume_raport
[Domeniu] [FOR expresie_logică1] [WHILE expresie_logică2]
[PREVIEW] [TO PRINTER]
```

Clauza `PREVIEW` se referă la previzualizarea raportului, iar clauza `TO PRINTER` se referă la listarea acestuia direct la imprimantă. Se observă că liniile care vor fi afişate pot fi filtrate direct din comanda `REPORT FORM` prin clauza `FOR`.

7.3. Rapoarte complexe

Am văzut mai înainte paşii care trebuie urmaţi pentru construirea unui raport „simpluţ". Cum în practică rapoartele simple sunt destul de rare, vom încerca să exemplificăm în continuare cum se poate construi un raport ce necesită câteva facilităţi mai avansate.

Presupunem că am primit o cerere pentru definirea unui raport despre situaţia curentă a facturilor. Făcând schiţa iniţială împreună cu beneficiarul raportului, deducem că:

- facturile trebuie grupate pe fiecare client;
- din raport trebuie să rezulte totalul vânzărilor pe fiecare client şi pe fiecare localitate;
- de asemenea, raportul trebuie să includă şi informaţia cu privire la restanţele la plată ale fiecărui client;
- paginile să fie numerotate, iar titlul raportului să apară doar pe prima pagină.

Structura de date se complică în special datorită informaţiei cu privire la restanţele de plată. Dar să abordăm problema gradual:

- informaţiile despre facturi implică tabelele FACTURI, LINIIFACT şi PRODUSE;
- informaţiile despre furnizor implică tabelele CLIENTI, LOCALITATI şi JUDETE;
- pentru a deduce restanţele fiecărui client vom avea în vedere tabelele INCASFACT, FACTURI, CLIENTI.

La baza raportului, pentru a simplifica într-o anumită măsură lucrurile, vom pune o tabelă temporară (cursor) obţinută în urma mai multor interogări. Astfel, pentru a obţine facturile şi informaţiile despre clienţi, vom efectua o interogare care va face *join*-urile tabelelor CLIENTI, LOCALITATI şi JUDETE pe de o parte, şi FACTURI, LINIIFACT şi PRODUSE pe de altă parte (vezi listingul 7.3).

Listing 7.3. Construirea cursorului cu detaliile despre facturi şi clienţii care le-au primit

```
SELECT c.codcl, c.dencl, c.codfiscal, c.adresa, c.telefon, L.loc, l.codpost, J.jud,;
  J.judet, F.nrfact, F.datafact, Lf.linie,;
  Lf.codpr, Lf.cantitate, Lf.pretunit, P.denpr,;
  P.um, P.proctva;
FROM  localitati l, clienti c, judete j,;
   facturi f, liniifact lf, produse p;
WHERE  P.codpr = Lf.codpr ;
  and  F.nrfact = Lf.nrfact ;
  and  C.codcl = F.codcl ;
  and  J.jud = L.jud ;
  and  L.codpost = C.codpost;
INTO CURSOR cfacturi;
ORDER BY l.codpost, c.codcl, f.nrfact, lf.linie
```

Se obsevă că rezultatul final al interogării anterioare este tabela temporară CFACTURI. În continuare vom efectua o altă interogare, din care să rezulte ce datorie are fiecare client, având în vedere că o factură este plătibilă în mai multe tranşe (vezi listingul 7.4).

Listing 7.4. Cursorul cu datoriile pe fiecare client

```
SELECT c.codcl, sum(i.transa) as platit FROM clienti c, facturi f, incasfact i;
WHERE i.nrfact = f.nrfact and c.codcl = f.codcl;
GROUP BY c.codcl INTO CURSOR ctranse
```

În final, vom uni cele două cursoare inițiale și vom obține astfel un cursor final din care se va forma raportul în sine. Cum există posibilitatea ca un client să nu-și fi plătit nici o factură (nu va apărea în cursorul CTRANSE), vom folosi joncțiunea exeternă de tip `LEFT OUTER JOIN` (vezi listingul 7.5).

Listing 7.5. Cursorul ce va constitui sursa de date a raportului

```
SELECT c1.*, nvl(t.platit, 0) as platit;
FROM cfacturi c1 LEFT JOIN ctranse t;
ON c1.codcl = t.codcl;
INTO CURSOR craport;
ORDER BY c1.codpost, c1.codcl, c1.nrfact
```

Toate aceste interogări vor fi incluse, în ordinea strictă în care au fost prezentate, în evenimentul `Init` al containerului `Data Environment`. Tot aici mai includem și următoarele instrucțiuni:

```
SET DATE TO BRITISH
SET CENTURY ON
```

prin care câmpurile de tip dată calendaristică ale raportului să primească datele în format european `DD/MM/YYYY (ZZ/LL/AAAA)` cu patru poziții pentru an.

7.3.1. Imprimarea preferențială pe anumite pagini

În continuare vom defini structura de afișare (formatul de listare) a raportului, încercând să-l facem să arate cât mai „profesional".

În primul rând ne vom ocupa de elementele cele mai simple: vom activa benzile `Title` și `Summary`, iar în banda `Title` vom crea o etichetă (un bloc text – `label`) care să consemneze titlul „Lista facturilor la data", lângă care vom insera un câmp cu data curentă.

Titlul raportului va apărea doar pe prima pagină. Pe celelalte pagini va apărea în antet un text care să amintească titlul raportului, iar în subsolul paginii va apărea numărul acesteia. Aceste informații trebuie tipărite pe toate paginile, în afară de prima. Cum se poate realiza acest lucru?

Întâi de toate, vom crea obiectele din antetul și subsolul paginii: blocul de text (`label`), câmpul de raport cu data curentă și câmpul de raport cu numărul paginii, care poate să conțină o expresie de genul `"Pag. "+allt(str(_pageno))`. După care se vor selecta toate aceste obiecte (cu tasta `SHIFT` apăsată și punctare cu mouse-ul pe fiecare dintre ele), iar din meniul contextual (click-dreapta ținând în continuarea tasta `SHIFT` apăsată) se selectează opțiunea `Properties`. Din fereastra astfel deschisă se acționează butonul `Print When...`, care va lansa la rându-i fereastra `Print When` (vezi figura 7.15). Apoi, în căsuța de text deasupra căreia se găsește eticheta `Print only when expression is true` se introduce expresia `_pageno > 1`, care va determina inhibarea listării respectivelor obiecte pe prima pagină.

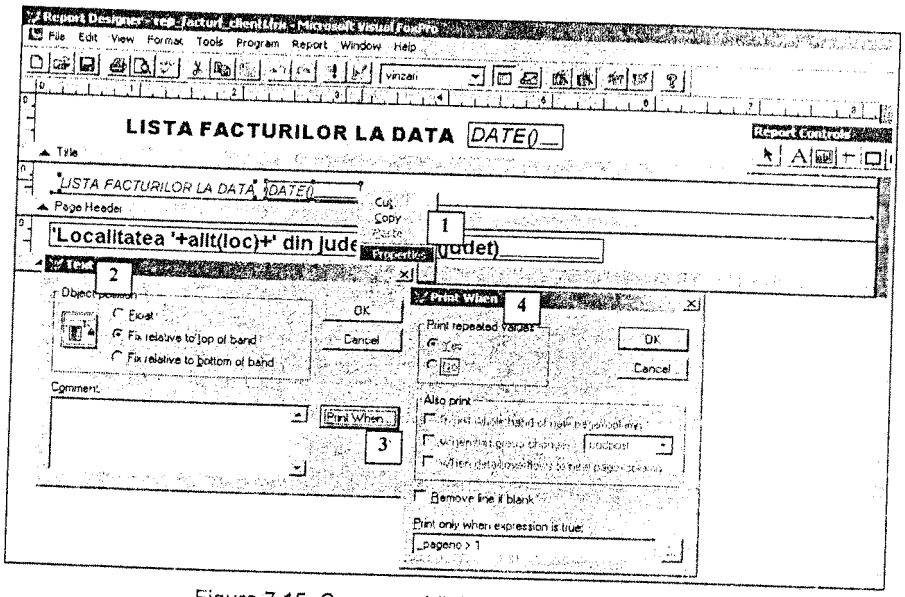


Figura 7.15. Cum se evită listarea pe prima pagină

Atunci când anumite obiecte din raport presupun un comportament sau caracteristici comune, pentru tratarea lor nediferențiată și totodată pentru creșterea productivității muncii de dezvoltare, aceste obiecte pot fi incluse într-o structură de grup. Acest lucru se realizează prin selectarea lor (așa cum am procedat mai înainte), după care, din meniul principal al VFP, se alege opțiunea `Format→Group`.

7.3.2. Gruparea înregistrărilor în raport

Cerințele raportului ne obligă să grupăm facturile pe fiecare client și să obținem totaluri privind vânzările pe client și localitate. În acest scop, vom forma un grup principal după localitate (mai exact, după codul poștal – `codpost`), un subgrup pentru fiecare client (după `codcl`), iar în cadrul acestuia, alt grup pentru liniile fiecărei facturi (după `nrfact`). Pentru ca raportul să se formeze corespunzător, *este obligatorie ordonarea datelor în sursa de date după aceleași câmpuri care vor forma expresiile de grupare din raport.* Așa se explică includerea clauzei `ORDER BY` în fraza SQL din care rezultă ultimul cursor – `CRaport`.

Într-un raport VFP grupurile se declară prin opțiunea `Report→Data Grouping...` din meniul pricipal VFP, care va lansa fereastra `Data Grouping`. În această fereastră se vor specifica în ordine expresiile după care se vor forma grupurile din raport (vezi figura 7.16).

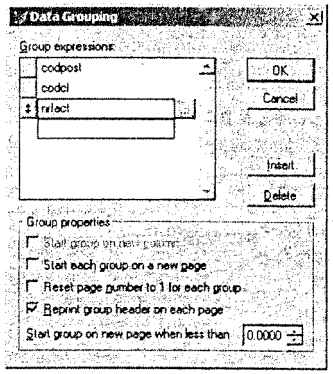


Figura 7.16. Declararea grupurilor din raport

În fereastra `Data Grouping` apar şi câteva opţiuni specifice care ne pot ajuta la dispunerea şi încadrarea mai raţională în pagină a liniilor care formează grupurile raportului. Astfel:

- grupurile care se formează după localităţi (după `codpost`) vor începe întotdeauna pe pagină nouă dacă se va bifa opţiunea `Start each group on new page`;
- antetul fiecărei facturi va fi reafişat eventual şi pe pagina următoare în care s-ar putea prelungi grupul, dacă se va bifa opţiunea `Reprint Group Header on each page`;
- de asemenea, pentru o afişare cât mai cursivă, grupurile formate pentru fiecare client vor putea fi afişate pe pagină nouă în cazul în care spaţiul până la limita inferioară a paginii este prea mic (se evită astfel situaţiile în care numele şi, eventual, adresa sunt afişate pe finalul unei pagini, iar informaţiile şi subgrupurile esenţiale sunt trimise pe pagina următoare). Acest lucru se realizează ajustând valoarea spinner-ului `Start group on new page when less than`.

În continuare vom crea în antetul grupului pentru localităţi un nou câmp, în care se va specifica numele localităţii şi judeţul în care se găseşte, folosind, pentru a respecta deziderатul de cursivitate a raportului, o expresie de genul `'Localitatea '+allt(loc)+' din judetul '+allt(judet)`. Apoi, în antetul subgrupului corespunzător fiecărui client se vor specifica:

- numele clientului, printr-un câmp de raport care va conţine expresia: `'Societatea '+allt(dencl)`, şi, eventual, codul fiscal, printr-un câmp care va conţine expresia `'Codul fiscal: '+allt(codfiscal)` (ambele informaţii sunt aliniate pe un singur rând);
- informaţii privind localizarea şi modul de contact al clientului respectiv, prin etichetele `Adresa` şi `Telefon` care însoţesc câmpurile în care sunt precizate expresiile `nvl(allt(adresa), 'necunoscuta')` şi `nvl(telefon, 'Indisponibil')`. Am folosit funcţia NVL pentru că aceste informaţii nu sunt întotdeauna specificate în baza de date.

7.3.3. Variabilele de raport

Antetul grupului corespunzător fiecărei facturi prezintă câteva particularităţi:

- în primul rând, va trebui specificat numărul şi data facturii, printr-un câmp ce va conţine expresia `'Factura Nr.'+allt(str(nrfact))+', din data '+allt(dtoc(datafact))`;
- în al doilea rând, trebuie construit un cap de tabel pentru detaliile fiecărei facturi, pe baza unor blocuri de text care vor consemna numele următoarelor coloane: `Linia, Produsul, Cantitatea, Pret Unitar, Valoarea fara TVA, Valoare Totala`, circumscrise de un obiect grafic de tip dreptunghi;
- în al treilea rând, apariţia facturilor din grupul unui client este numerotată folosind o variabilă de raport.

Pentru a numerota apariţia fiecărei facturi în grupul de clienţi, vom declara două variabile de raport `NrFact_` şi `Nr_Crt_` (vezi figura 7.17).

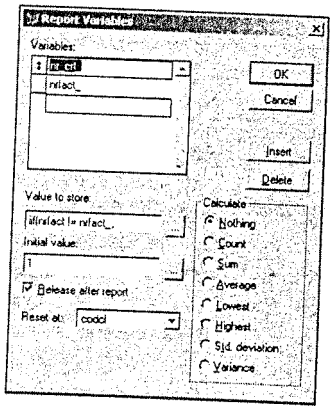


Figura 7.17. Declararea unei variabile de raport

Specificarea unei variabile de raport presupune următoarele: precizarea numelui acesteia, a valorii iniţiale, a unei expresii care va determina valoarea efectivă ce se va stoca şi actualiza pentru fiecare înregistrare sau linie din raport, a momentului în care va fi resetată la 0 sau la valoarea iniţială şi, de asemenea, dacă va fi eliberată din memorie după închiderea raportului (vezi opţiunile din fereastra prezentată în figura 7.17).

Motivul pentru care am folosit două variabile este următorul: `Nr_Crt_` este cea care va fi afişată efectiv, iar `NrFact_` este o variabilă de control, verificarea acesteia marcând momentul în care se trece la o nouă factură, `Nr_Crt_` fiind incrementat cu 1. La definirea celor două variabile s-au precizat:

- pentru `NrFact_`:
  - Valoarea iniţială: `Nrfact`;
  - Valoarea de stocare: `Nrfact`;
  - Momentul resetării: sfârşitul raportului (*end of report*);
- pentru `Nr_Crt_`:
  - Valoarea iniţială: 1;

- Valoarea de stocare: expresia `iif(nrfact!=nrfact_, nr_crt_+1,nr_crt_)` – cu alte cuvinte, incrementarea valorii curente dacă se ajunge în înregistrările unei noi facturi;
- Momentul resetării: `codcl`, adică sfârșitul grupului corespunzător clientului curent.

Revenind în antetul grupului corespunzător fiecărei facturi, pentru numerotarea facturii curente se creează un câmp de raport care va conține expresia `allt(str(nr_crt_))+')'`.

7.3.4. Introducerea câmpurilor calculate

În banda de detaliu a raportului vor fi incluse câmpurile care vor consemna informațiile privind numărul liniei din factura curentă (câmpul `linie`), denumirea produsului din linia curentă (`DenPr`), prețul unitar al produsului din linia curentă (`PretUnit`). De asemenea, vor mai fi introduse două câmpuri suplimentare, care ar trebui să calculeze pentru fiecare linie:

- *valoarea fără TVA*, după expresia `Cantitate*PretUnit`;
- *valoarea totală*, după expresia `(1+ProcTVA)*(Cantitate*PretUnit)`.

În benzile de final ale grupurilor din raport sunt plasate diferite informații sintetice. Astfel, în subsolul grupurilor corespunzătoare fiecărei facturi vom calcula totalul cu și fără TVA al acestora. În acest scop, se vor crea două câmpuri calculate după expresiile consemnate anterior (cele din secțiunea de detalii). Apoi, din fereastra `Expression` se acționează butonul `Calculations...`, iar din fereastra `Calculations` ce se va deschide astfel, se selectează butonul radio `Sum` din cadrul `Calculate`. În același timp se observă că în lista combinată `Reset` este deja selectată expresia grupului corespunzătoare facturilor – `nrfact` (în caz contrar, se va alege manual opțiunea `NrFact` din acestă listă).

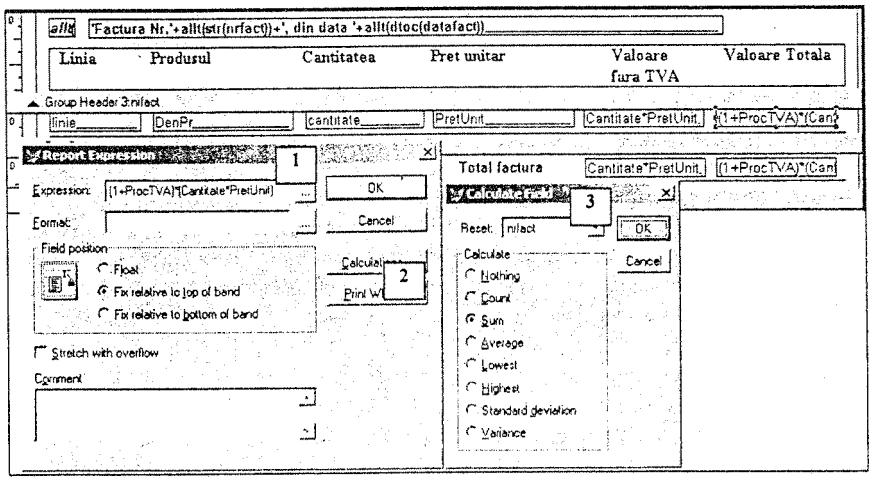


Figura 7.18. Declararea câmpurilor ce conțin calcule sintetice

La nivelul grupului corespunzător fiecărui client (mai exact, în subsolul acestui grup) ne interesează:

- valoarea totală a vânzărilor și afișarea nivelului plăților;
- eventualele restanțe.

Valoarea totală a vânzărilor se obține printr-un câmp ce face un calcul de tip `Sum` pentru expresia `(1+ProcTVA)*(Cantitate*PretUnit)` la nivelul fiecărui client, iar totalul plăților se poate obține printr-un câmp de raport care va afișa valoarea din câmpul `Platit` din cursorul de bază (nivelul plăților pe fiecare client a constituit motivul pentru care am construit fraza `SELECT SQL` din care rezultă cursorul `CTranse`).

Afișarea eventualelor restanțe e o problemă ceva mai delicată. Primul impuls ar fi crearea unui câmp agregat (calcul de tip `Sum`) la nivelul grupului corespunzător clientului după expresia `(1+ProcTVA)*(Cantitate*PretUnit)-Platit`. Din păcate, în acest fel va fi agregat și câmpul `Platit`, ceea ce va denatura rezultatul dorit. Pentru a respecta acuratețea datelor, vom crea variabila `de_plata`, al cărei scop este efectuarea calculului agregat al vânzărilor pe fiecare client. Pentru această variabilă se va consema ca expresie de stocare `(1+ProcTVA)*(Cantitate*PretUnit)`, iar din cadrul `Calculate` al ferestrei de specificare a variabilelor se va alege `Sum` (vezi figura 7.19). În fine, revenind în subsolul grupului corespunzător fiecărui client, se va crea un câmp care va conține expresia `iif(de_plata-platit>0,de_plata-platit,0)`.

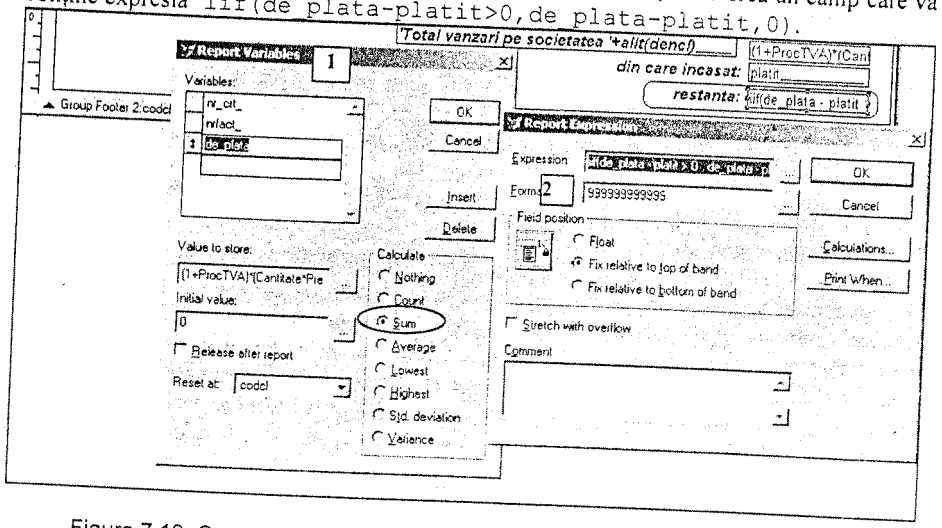


Figura 7.19. Consemnarea unui câmp calculat care invocă o variabilă de raport

Eventual, dacă se dorește afișarea restanțelor numai dacă acestea există, atunci în căsuța `Print only when expression is true` din fereastra `Print When` (vezi paragraful despre tipărirea preferențială) se va preciza expresia `de_plata-platit>0`.

Totalul vânzărilor pe fiecare localitate se rezolvă printr-un câmp agregat plasat în subsolul grupului corespunzător fiecărei localități (format după `codpost`) care să conțină expresia `(1+ProcTVA)*(Cantitate*PretUnit)`.

7.3.5. Formatul de afişare pentru datele numerice

Pentru ca toate câmpurile ce consemnează valori să fie afişate în acelaşi mod, se recomandă o formatare uniformă folosind opţiunea `Format` din fereastra `Report Expression`. De exemplu, pentru a afişa toate câmpurile ce consemnează valori în format numeric, fără zecimale, cu maxim 12 poziţii pentru întregi, se selectează butonul din stânga căsuţei de text `Format`, apoi, din fereastra ce se va deschide, se selectează butonul radio `Numeric`, iar în căsuţa de text se consemnează formatul `999999999999`.

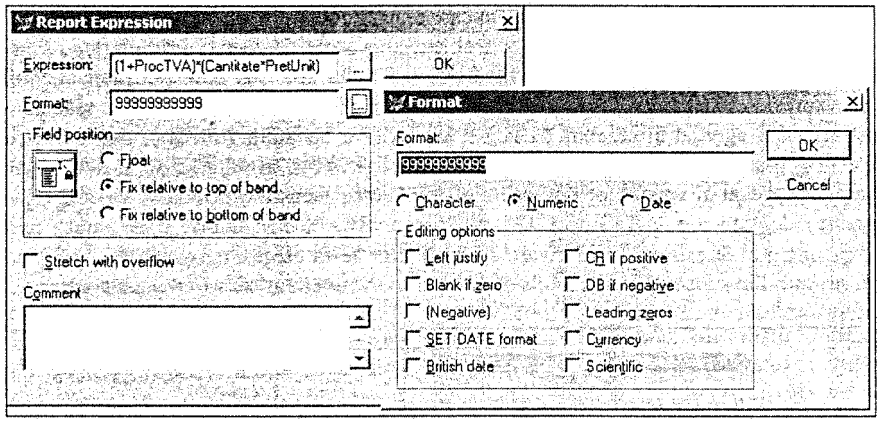


Figura 7.20. Stabilirea formatului numeric pentru câmpurile ce consemnează valori

7.3.6. Alinierea obiectelor în raport

O bună parte din timpul realizării unui raport se consumă pentru a aşeza, cât mai estetic, obiectele în raport. Există însă o serie de opţiuni specifice care simplifică munca de aliniere, centrare, redimensionare etc. a obiectelor din raport. Spre exemplu, dacă dorim ca toate câmpurile care consemnează sumele de pe coloana `Valoare totală` din raport să aibă aceeaşi dimensiune şi să fie aliniate în acelaşi sens (formatarea numerică uniformă am exemplificat-o anterior), atunci toate acestea se selectează în acelaşi timp (ţinând, bineînţeles, tasta `SHIFT` apăsată), după care se apelează la opţiunile din submeniul `Format`:

- `Size→To Widest` – pentru a le redimensiona luând ca referinţă dimensinea celui mai larg;
- `Size→To Narrowest` – pentru a le redimensiona luând ca referinţă dimensinea celui mai îngust;
- `Align→Left Side` – pentru a le realinia în funcţie de cel mai din stânga;
- `Align→Right Side` – pentru a le realinia în funcţie de cel mai din dreapta.

Acest opţiuni, la fel ca şi bara de instrumente `Report Layout`, conţin şi alte posibilităţi de ajustare a plasamentului obiectelor sau structurii de afişare.

În figurile 7.21 şi 7.22 este prezentată macheta finală a raportului în faza de proiectare şi de previzualizare.

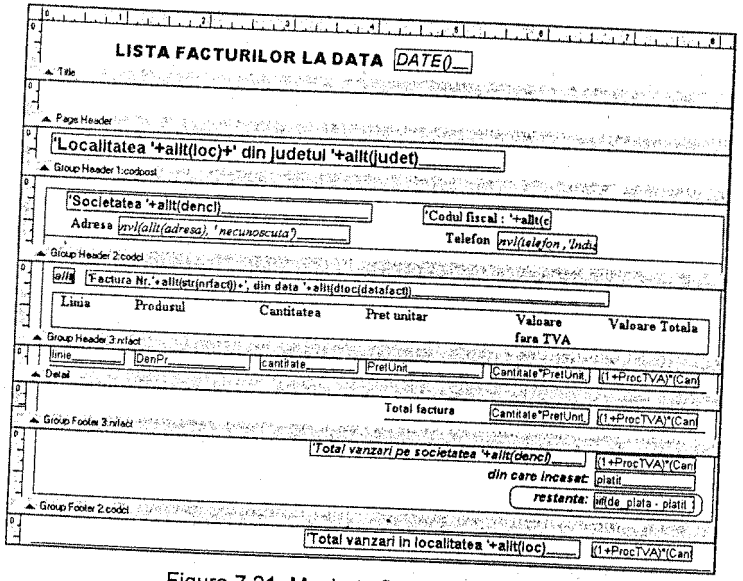


Figura 7.21. Macheta finală a raportului

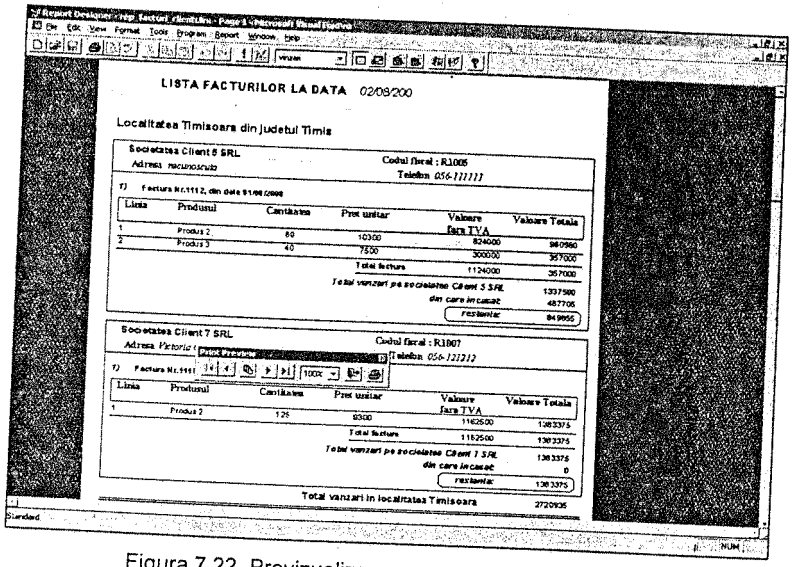


Figura 7.22. Previzualizarea raportului în forma finală

7.4. Setarea imprimantei la momentul execuției. Modalități de imprimare rapidă

După cum am văzut anterior, imprimarea sau previzualizarea raportului se poate realiza utilizând sintaxa:

```
REPORT FORM NumeRaport [TO PRINTER] [PREVIEW] [PROMPT]
      [RANGE nStartPage [, nEndPage]] [SUMMARY]
```

Parametrii pe care dorim să-i evidențiem aici sunt:

- `PROMPT` – care va deschide o fereastră pentru setarea opțiunilor de imprimare înainte de a trimite raportul la imprimantă (figura 7.23). Poate fi utilizat numai în combinație cu parametrul `TO PRINTER`;
- `RANGE nStartPage [, nEndPage]`– care va trimite la imprimantă numai paginile cuprinse în intervalul specificat prin numerele de început și de sfârșit;
- `SUMMARY` – care *nu va imprima linia de detaliu* a raportului, obținându-se astfel numai liniile de grupare, titlul, sumarul, antetul și piciorul de pagină. Utilizarea acestui parametru este utilă atunci când dorim să construim un singur raport, pentru a obține o situație atât centralizată, cât și detaliată, după caz.

Pentru a furniza utilizatorului final al unei aplicații posibilitatea de setare a imprimantei, în funcție de necesități, putem apela și la o funcție ce furnizează informații foarte utile preluate de la sistemul de operare: `SYS()`. Spre exemplu, `SYS(1037)` va deschide fereastra de dialog specifică pentru setarea imprimantei, iar `SYS(13)` va returna un șir de caractere reprezentând starea curentă a imprimantei (`READY`, `OFFLINE`).

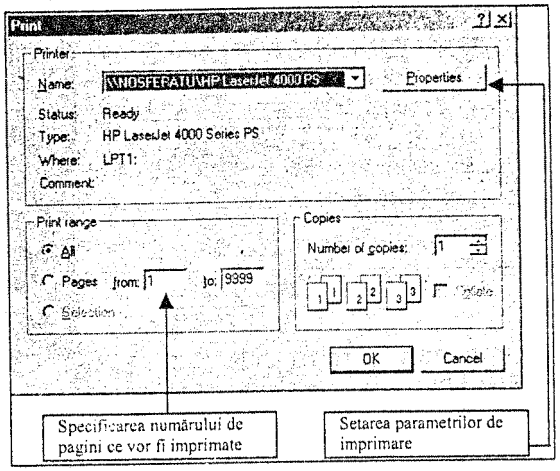


Figura 7.23. Rezultatul utilizării parametrului `PROMPT` în comanda de imprimare a unui raport

Viteza de imprimare a raportului final constituie un aspect extrem de important atunci când purcedem la proiectarea acestuia. Ea depinde, bineînțeles, de performanțele imprimantei în primul rând, dar și de *stilul de lucru* în proiectarea rapoartelor. Astfel, cu cât vom folosi formate mai complexe pentru obiecte (fonturi mai multe, caractere îngroșate etc.), cu atât viteza de imprimare scade, deși raportul final are un aspect mai îngrijit. Dimpotrivă, renunțarea la aspect în favoarea unui grad de funcționalitate sporit poate constitui un imperativ, în special în cazul aplicațiilor economice care, în funcție de mărimea firmei, pot genera „mormane" de listinguri într-o perioadă scurtă de timp (de exemplu, închiderea contabilă a lunii), de multe ori pe imprimante matriceale.

Pentru **sporirea vitezei de imprimare** există posibilitatea de a utiliza driver-ul de imprimantă numit `Generic/textonly`, pe care îl găsim în orice sistem de operare Windows 9x/NT/2000. Acesta poate fi instalat din meniul principal al Windows, `Start→Setings→Printers→Add Printer`. Dezavantajul acestuia constă în faptul că nu „știe" să imprime decât caractere ASCII, un alt fel de obiect (linii, imagini, stiluri de formatare a fonturilor) fiind ignorat. De asemenea, nu poate fi utilizată formatarea câmpurilor în raport. În schimb, are ca avantaj o viteză de imprimare foarte mare pe imprimante matriceale.

7.5. Rapoarte dinamice în VFP

Orice dezvoltator de aplicații cu ceva experiență s-a întâlnit, măcar o dată, cu cerința de a obține un raport ale cărui dimensiuni pe orizontală variază în timp atât ca număr de câmpuri, cât și, eventual, ca denumire a acestora. Cu alte cuvinte, în momentul proiectării raportului se cunoaște doar forma sa generală (aranjare în pagină, eventuale subtotaluri pe grupuri și totaluri generale), dar nu se cunoaște cu exactitate numărul de câmpuri și/sau denumirea acestora. Această stare de fapt induce necesitatea construirii unei *structuri dinamice*, de tip „acordeon", care să furnizeze în final informații în funcție de situația existentă în baza de date, nimic în plus sau în minus.

Exemplul cel mai concludent pentru un raport dinamic este cel referitor la „verticalizarea înregistrărilor". Vom recurge la un exemplu similar cu cel din capitolul 5, subcapitolul referitor la SQL și macrosubstituție: se dorește obținerea unei situații care să cuprindă totalul valoric facturat clienților, defalcat pe produse cu total pe fiecare client în parte (cu alte cuvinte, cine, ce și cât a cumpărat), situația trebuind să fie prezentată în maniera din figura 7.24. Nu este, poate, cel mai bun exemplu, având în vedere volatilitatea înregistrărilor din tabela CLIENTI (ar fi fost mai nimerită, poate, o „verticalizare" pe gestiuni).

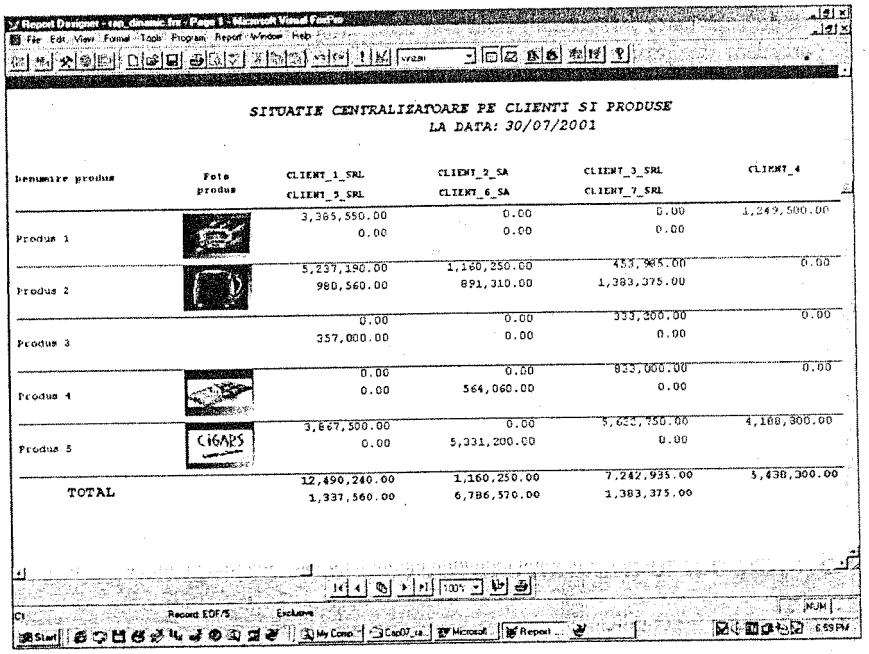


Figura 7.24. Raportul final (la previzualizare) pentru exemplul privind verticalizarea

Având în vedere faptul că numele clienților trebuie să constituie capul de tabel, iar aceştia se regăsesc în baza de date sub formă de linii (înregistrări), va trebui:

- în primul rând, să construim o sursă de date mobilă (în funcție de câți clienți avem la un moment dat, să obținem tot atâtea coloane). Pentru aceasta vom recurge, bineînțeles, la un cursor (tabelă temporară) – codul fiind exemplificat în listingul 7.6.
- în al doilea rând, vom proiecta macheta raportului astfel încât câmpurile definite în raport să fie vizibile sau nu, în funcție de existența sau nu a coloanelor corespunzătoare în cursor (figura 7.25).

Listing 7.6. „Verticalizarea" înregistrărilor

```
*o varianta de verticalizare

v_fraza_select='select denpr,codpr'
IF !USED('clienti')
    USE clienti IN 0
ENDIF
SELECT clienti
* !* vom "construi" fraza select SQL
SCAN
    v_client=f_inlocuieste(ALLTRIM(clienti.dencl))
    v_cod=clienti.codcl

    v_fraza_select=v_fraza_select+",f_calcul_("+ALLTRIM(STR(v_cod))+",codpr) as "+v_client
    v_fraza_union=v_fraza_union+",f_calcul_("+ALLTRIM(STR(v_cod))+")"

ENDSCAN
```



```
v fraza select=v fraza select+ ' from produse into cursor c1'

* ! * şi in final macrosubstitutia:

&v_fraza_select

FUNCTION f_inlocuieste          && necesară pentru inlocuirea spatiilor din numele clientului
                                && cu un caracter valid pentru nume de campuri
    PARAMETERS p_sir,p_chr
    v_sir_nou=""
    FOR i=1 TO LEN(p_sir)
        IF SUBSTR(p_sir,i,1)=' '
                v_sir_nou=v_sir_nou+'_'
        ELSE
                v_sir_nou=v_sir_nou+SUBSTR(p_sir,i,1)
        ENDIF
    ENDFOR
    RETURN v_sir_nou
END FUNCT

FUNCTION f_calcul_              && calculează totalul facturat pe produs cumparat

    PARAMETERS p_client,p_codpr
    DIMENSION a_total(1)
    a_total(1)=0
        SELECT SUM(cantitate*pretunit*(1+proctva));
                FROM liniifact l,facturi F,produse p;
                WHERE l.nrfact=F.nrfact AND p.codpr=l.codpr ;
                AND F.codcl=p_client AND p.codpr=p_codpr;
                INTO ARRAY a_total


RETURN a_total(1)
END FUNCT
```

Observăm în figura 7.25 că modalitatea clasică de realizare a antetului paginii în raport, precum şi definirea câmpurilor în zona de detaliu a raportului a fost înlocuită cu utilizarea unei expresii generale. Astfel:

- în antetul de pagină (acolo unde trebuie să apară numele fiecărui client avem o expresie de genul: `IIF(A_NumarCampuri>=x,field(x+3),'');`
- în zona de detaliu (unde trebuie să apară efectiv valoarea de pe fiecare înregistrare a câmpului din antet) utilizăm o expresie de genul: `IIF(A_NumarCampuri>=x,valoare(field(x+3)),0);`
- în sfârşit, în zona de sumar introducem aceeaşi expresie, dar cu opțiunea de însumare la finalul raportului.

Valoarea *x* reprezintă de fapt numărul de ordine al câmpurilor cu referire la clienți, definite fizic în raport.

Variabila globală `A_NumarCampuri` utilizată în expresiile de mai sus este inițializată în procedura de creare a cursorului sursă de date cu o valoare egală cu numărul de clienți din baza de date. Ea se comportă ca un contor ce va determina numărul de câmpuri tipăribile în raportul final. Aşadar, dacă `a_NumarCampuri` este mai mare sau egal cu o valoare *x*, înseamnă că avem cel puțin *x* clienți de afişat şi, ca urmare, cel puțin *x* câmpuri ce vor avea valori în raportul final.

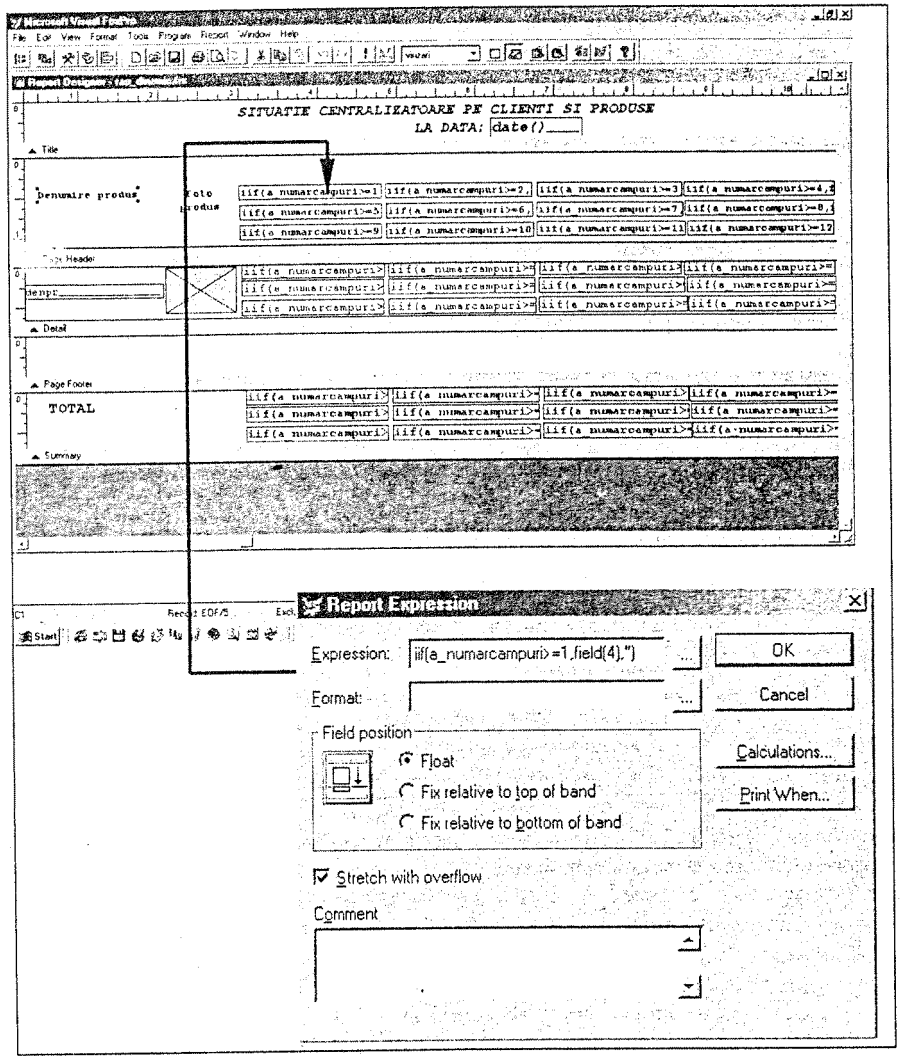


Figura 7.25. Exemplu de proiectare a unui raport dinamic

Pentru că în cursorul-sursă (`C1`) numele clienților îl găsim sub formă de nume de câmp, utilizăm funcția `FIELD(NumarCamp)`, ce returnează numele câmpului specificat prin `NumarCamp` din tabela curentă (numărătoarea începe cu 1 de la stânga la dreapta), pentru a afișa numele clienților în antetul de pagină. Se observă că numărul câmpului cu care este apelată funcția este diferit de valoarea de comparație a variabilei `A_Numar_Campuri`, diferența fiind de 3 unități. Această diferență decurge din faptul că primul client din tabela CLIENTI se va regăsi, în cursorul `C1`, pe coloana a patra, primele trei câmpuri ale cursorului fiind numele produsului, codul său și imaginea asociată.

Funcția `VALOARE(NumeCâmp)` va returna valoarea câmpului specificat ca parametru de pe înregistrarea curentă din cursorul `C1`. Ea este utilizată în expresia din zona de detaliu pentru a obține valorile câmpului curent. Conținutul ei îl regăsim în listingul 7.7 (de observat macrosubstituția `&pcamp`):

Listing 7.7. Definiția funcției `Valoare()`

```
PARAMETERS pcamp
    v_valcamp=0
    v_valcamp=&pcamp
RETURN v_valcamp
```

În sfârșit, funcția `IIF()` va returna numele câmpului (în antetul de pagină) și valorile lui (în zona de detaliu), atât timp cât regăsim atribute corespunzătoare în cursorul `C1`, deci cât timp `A_NumarCampuri` >=x (unde x reprezintă de fapt numărul câmpului referitor la clienți definit în raport). În momentul în care numărul câmpului definit fizic în raport este mai mare decât numărul de clienți din baza de date, funcția va returna valoarea 0 pentru zona de detaliu și o valoare vidă, specificată prin două apostrofuri unul după altul (''), pentru zona antetului de pagină (unde ar trebui să avem numele clientului).

Privind comparativ figura 7.25 (faza de proiectare a raportului) și figura 7.24 (faza de previzualizare a raportului final), observăm că:

- în faza de proiectare avem definite douăsprezece câmpuri pentru extragerea informațiilor referitoare la clienți, în timp ce în raportul final sunt vizibile (și deci imprimabile) doar acelea care și-au găsit câmp corespondent în cursorul `C1` (numai șapte dintre ele);
- dacă toate câmpurile de pe o linie afișează „nimic" sau 0, spațiul alocat întregii linii va fi suprimat din raportul final (linia liberă din zona de detaliu care apare în figura 7.24 se datorează dimensiunilor câmpului general ce are ca sursă imaginea produsului);
- numărul maxim de câmpuri din raportul final depinde, pe de o parte, de numărul de atribute de care dispune sursa de date, iar pe de altă parte, de numărul de câmpuri de raport definite la construirea acestuia. În exemplul nostru am definit numai douăsprezece câmpuri de raport cu referire la clienți, ceea ce va limita, bineînțeles, afișarea a maxim doisprezece clienți, indiferent câți ar fi în baza de date.

Conform notațiilor din figura 7.26, elementele necesare implementării unui comportament mobil obiectelor din raport, din punctul de vedere al așezării în pagină, ar fi:

1. Obiectul (în cazul nostru câmpul) curent va fi afișat, la previzualizare sau imprimare, numai dacă prin evaluarea expresiei obținem adevărat;
2. Dacă nu mai există un alt obiect pe aceeași linie care să poată fi imprimat sau vizualizat, întreaga linie (pe care se găsește obiectul curent) va fi ștearsă din raportul final, iar spațiul ocupat inițial va fi redistribuit obiectelor de pe liniile următoare;
3. Obiectul curent își va modifica poziția (la previzualizare sau imprimare), față de cea definită în momentul proiectării raportului, în funcție de noua poziție a obiectelor înconjurătoare;
4. Obiectul curent își va modifica dimensiunea în funcție de dimensiunea sursei de date pe care trebuie să o afișeze. Setarea este aplicabilă numai obiectelor de tip `Field (câmp)`;
5. Expresia ce definește sursa de date a câmpului va afișa „nimic" în cazul în care o condiție anume nu este îndeplinită.

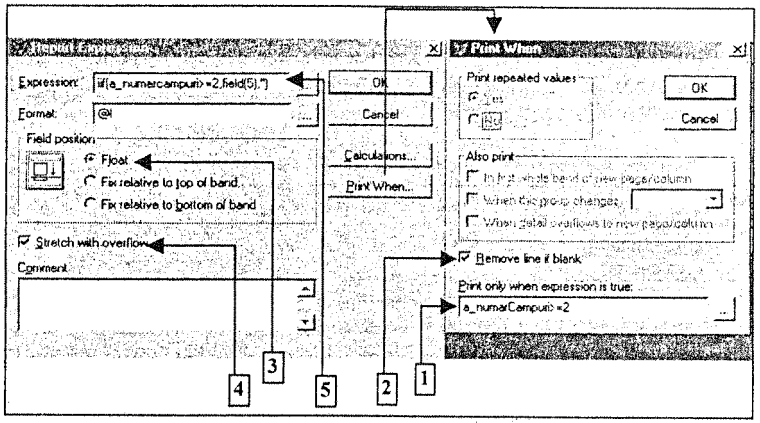


Figura 7.26. Setări specifice pentru definirea mobilității obiectelor în raportul final față de pozițiile lor inițiale în faza de proiectare

Ne vom referi în final și la coloana „Foto" din raport, ce cuprinde un câmp de tip `Picture/OLE bound control` pentru care sursa de date o constituie câmpul general `Imagine` din același cursor `C1`. Setările necesare unui astfel de câmp le regăsim în figura 7.27:

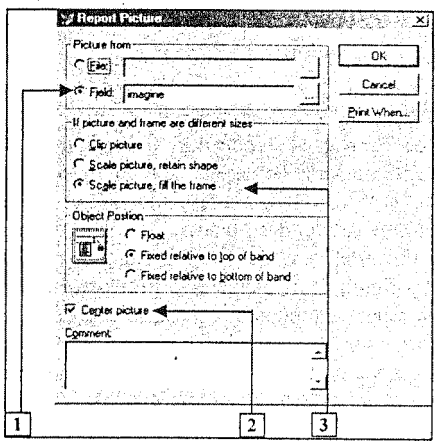


Figura 7.27. Setarea opțiunilor unui câmp de tip `Picture`

1. Specifică faptul că sursa de date este constituită de un câmp al tabelei curente (în cazul nostru, cursorul `C1`);
2. Centrează imaginea preluată din câmpul general în cadrul obiectului definit în raport, dacă imaginea este mai mică decât dimensiunile acestuia;
3. Redimensionează imaginea-sursă până la umplerea completă a scheletului definit prin obiectul din raport, chiar dacă aceasta presupune distorsiuni pe orizontală sau verticală.

Capitolul 8
Formulare Visual FoxPro

Realizarea formularelor (*forms*) în Visual FoxPro (și nu numai) reprezintă partea cea mai „spumoasă", dar și cea mai dificilă a dezvoltării de aplicații. Utilizăm formulare pentru a construi o interfață-utilizator prietenoasă și atractivă, deși nu putem spune că o aplicație bună se rezumă la o interfață spectaculoasă. Rolul formularelor este capital în preluarea datelor. Văzute „de departe", formularele sunt niște ferestre care pot conține *obiecte* ce răspund printr-un anume *comportament* (implementat prin *metode*) la *evenimente* declanșate de utilizator, de sistem sau de alte obiecte, constituind și *o modalitate de implementare efectivă a claselor noi* definite de dezvoltatori. Cu obiectele unui formular ne vom întâlni ceva mai târziu, pe parcursul acestui capitol, iar clasele definite de utilizator le vom aborda în capitolul 11.

8.1. Generalități. Tipuri de controale

Privite „mai de aproape", formularele sunt niște module de program complicate. Dar nu intrați în panică... doar câteva secvențe de cod vor fi scrise efectiv (folosind ceea ce s-a prezentat în capitolul 4), altminteri atributul „Visual" din numele mediului Visual FoxPro nu și-ar mai avea rostul.

În primele versiuni FoxPro, crearea unui ecran de introducere a datelor era o sarcină grea, necesitând combinarea mai multor comenzi din categoria `@SAY...GET...READ` (le-am menționat în capitolul 4) ce desemnează regiuni de afișare a datelor și, respectiv, regiuni capabile să primească date de la tastatură ori click-uri de la mouse. Chiar de la versiunea 2.0 s-a introdus un asistent care permite crearea unui formular de o manieră vizuală, prin trasarea obiectelor pe reprezentarea unei ferestre, urmată de scrierea manuală a procedurilor-eveniment (sau metodelor), sintagme prin care se desemnează codul-sursă „responsabil" de funcționarea formularului. Pe atunci era accesibil programatorului și codul-sursă „însărcinat" cu plasarea acelor regiuni pe ecran, la coordonate precise, cod care este ascuns în versiunile pentru Windows (unde mecanismul afișării este mult mai complex). În Visual FoxPro, deși modul de lucru s-a păstrat în mare parte, programatorul nu mai vede codul-sursă necesar afișării, ci doar acela necesar tratării evenimentelor. Putem crea un formular prin una din următoarele metode:

- în `Project Manager` selectăm cadrul de pagină `Forms` și efectuăm click pe butonul `New`;
- în meniul principal alegem `File→New...` (sau butonul `New` din bara principală de instrumente) și în ecranul apărut alegem opțiunea `Form`;
- în fereastra `Command` scriem comanda `CREATE FORM` (eventual cu numele fișierului și calea pe disc unde se va memora formularul). Fiindcă veni vorba, acest fișier are extensia `.SCX` și este însoțit întotdeauna de un fișier cu același nume și extensia `.SCT`.

Notă. Numai prima metodă va adăuga, în mod implicit, formularul la proiectul general al aplicației. Pentru celelalte două modalități este necesară o operațiune suplimentară de adăugare prin intermediul butonului ADD din Project Manager.

Ca urmare a uneia dintre acțiunile de mai sus, vom obține o fereastră ce reprezintă formularul care urmează a fi populat cu obiecte, încadrată în fereastra mai mare a Form Designer-ului. Totodată, sunt vizibile următoarele bare de unelte (figura 8.1): Form Designer, Form Controls, Layout și Color Palette. Cea mai utilizată în procesul dezvoltării va fi probabil bara Controls. În cazul în care una dintre aceste bare nu apare la deschiderea instrumentului Form Designer, o putem activa prin selectarea opțiunii View|Toolbars din meniul principal.

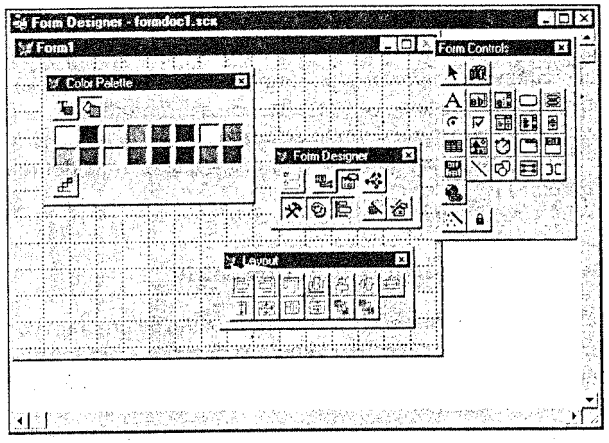


Figura 8.1. Proiectantul de formulare (Form Designer) și barele de unelte

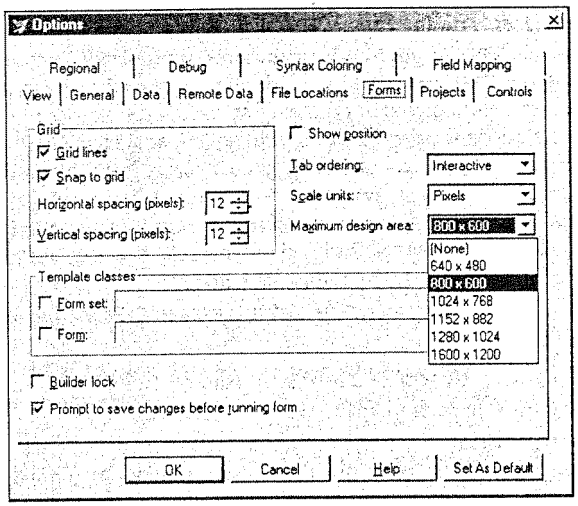


Figura 8.2. Stabilirea dimensiunilor maxime ale formularelor

Orice formular, asemenea tuturor componentelor unui proiect Visual FoxPro, se salvează într-un fișier și este bine să ne decidem de la început asupra unui nume, iar primul pas să fie chiar salvarea. Putem și amâna această operație. În orice caz, folosiți un nume sugestiv și nu denumirea implicită Formdoc1, ce poate naște confuzii (o vedeți pe bara de titlu cea mai de sus din figura 8.1 – nu o confundați cu linia de titlu inferioară, care arată altceva, anume proprietatea Caption a formularului, la care vom reveni).

Există o chestiune importantă pe care trebuie să o avem în vedere la proiectarea formularelor, și anume *care va fi rezoluția monitorului pe care va rula formularul.* Răspunsul la această întrebare trebuie cunoscut înainte de a ne apuca de lucru, pentru a nu constata la sfârșit, când totul pare bine și frumos pe calculatorul pe care a fost realizată interfața, că pe stația utilizatorului formularele noastre nu încap pe ecran. O dată cunoscută rezoluția de lucru, vom purcede la setarea dimensiunilor maxime ale formularului prin utilizarea opțiunii de meniu: Tools→Options, pagina Forms (figura 8.2). Pentru PC-urile contemporane cu ecrane de 15-17 inch în diagonală, rezoluția de 800×600 de pixeli este recomandabilă. Se poate merge și la 1024×768, dar țineți cont că anumite detalii pot deveni greu de observat și obosesc ochiul. Consultați-vă totuși cu beneficiarul, dacă aveți posibilitatea – s-ar putea ca sistemul lui să aibă alte performanțe, mai bune sau mai slabe decât al dvs.

Atenție! Dacă ați modificat dimensiunile *form*-ului, veți constata că ele nu intră imediat în vigoare. Trebuie închisă fereastra Form Designer (eventual cu salvarea modificărilor) și redeschisă, lucru posibil prin comanda File→Open..., prin selectarea formularului respectiv în fereastra corespunzătoare a Project Manager-ului și acționarea butonului Modify sau prin comanda MODIFY FORM <cale_și_nume_formular>. Dacă nu am reținut calea, putem scrie comanda MODIFY FORM ?, care va afișa dialogul de tip Open. Un dreptunghi alb pe ecran va indica întinderea maximă pe care o poate lua formularul.

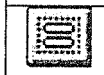
Pe un formular putem adăuga următoarele *tipuri de obiecte*:

- controale standard (bara de unelte Controls);
- containere (controale de un tip special, care conțin alte controale standard sau nu);
- clase definite de utilizator (vom vedea aceasta în capitolul 11);
- controale din categoria ActiveX (veți afla mai multe tot în capitolul 11).

Scopul acestui capitol este de a vă iniția în realizarea de formulare simple, fără efortul mai consistent pe care-l presupun diferite efecte vizuale deosebite ori anumite generalizări ale formularelor.

Tabelul 8.1. prezintă toate obiectele ce pot fi utilizate în construirea unui formular. Simbolul asociat fiecărui obiect în bara de unelte Controls este redat în coloana din stânga.

Tabelul 8.1. Simbolurile barei de unelte `Controls`

Simbolul butonului	*Denumire*	*Funcționalitate*	*Poate fi legat la o sursă de date*
	`Select Objects` (Selector) NU este un control	Selectează obiecte, în vederea redimensionării și/sau mutării lor. După crearea unui obiect, acest buton este selectat automat. Selecția se face prin click pe obiect, click cu tasta `Shift` apăsată dacă e nevoie a selecta mai multe obiecte.	NU
	`Label` (Etichetă)	Obiect de tip etichetă care afișează un text. Se folosește în general pentru a specifica titlul sau alte informații despre celelalte obiecte.	NU
	`Text Box` (Casetă[26] de text)	Afișează și dă posibilitatea modificării unei surse de date formate din valoarea unui câmp al unei tabele sau dintr-o variabilă de memorie. Afișează o singură linie de text (chiar dacă pe mai multe rânduri, este totuși o singură linie – nu se poate insera alta nouă cu `Enter`).	DA
	`Edit Box` (Casetă de editare text)	Are cam aceeași funcționalitate ca și controlul `Text Box`, doar că afișează sursa de date pe mai multe linii (se poate începe linie nouă cu `Enter`). Folosit în general pentru a introduce fragmente mai mari de text (câmpuri de tip memo, de exemplu).	DA
	`Command Button` (Buton de comandă)	Obiect care simulează vizual un buton al unui aparat oarecare (poate fi „apăsat"). În general răspunde la un eveniment `Click`. Folosește la declanșarea unor operații.	NU
	`Command Group` (Grup de butoane de comandă)	Container ce conține un număr variabil de butoane de comandă, fiecare având propria funcționalitate. Avantajul constă în faptul că pot fi manipulate unitar (ca un singur obiect) în funcție de necesități.	NU
	`Option Group` (Grup de butoane de opțiune)	Container ce conține un anumit număr de butoane cu aspect circular, fiecare însoțit de un text explicativ. Permite utilizatorului alegerea *unei singure opțiuni din cele posibile la un moment dat*. Mai sunt cunoscute și sub numele de *butoane-radio*.	DA
	`Check Box` (Casetă de opțiune)	Oferă utilizatorului posibilitatea de a opta între *adevărat* sau *fals* pentru o anumită condiție. Spre deosebire de butoanele de opțiune, grupurile de obiecte de tip „bifă" pot oferi posibilitatea *alegerii mai multor opțiuni din cele posibile*.	DA

26. În loc de „casetă" vom folosi și termenul „căsuță".

Simbolul butonului	*Denumire*	*Funcționalitate*	*Poate fi legat la o sursă de date*
	`List Box` (Casetă cu listă)	Obiect ce afișează o listă de elemente în care utilizatorul se poate deplasa pe verticală și poate selecta unul sau mai multe la un moment dat.	DA
	`Combo-box` (Listă combinată)	Obiect asemănător cu o casetă de text, ce deschide (la apariția unui eveniment – de exemplu `Click`) o listă de elemente din care utilizatorul poate alege una singură. După selecția necesară lista se închide. Foarte util pentru economisirea spațiului pe formular. Partea superioară (vizibilă tot timpul) permite introducerea de text pentru căutare în listă ori pentru adăugarea de elemente noi, după caz.	DA
	`Spinner`	Afișează într-o căsuță de text un contor (număr) ce poate lua valori între anumite limite. Incrementarea, respectiv decrementarea au loc prin click pe săgețile sus-jos sau prin scrierea unui număr de la tastatură.	DA
	`Grid`	Obiect de tip container, deosebit de complex, ce poate afișa și manipula datele într-o formă tabelară (ca o foaie de calcul Excel). Poate include alte tipuri de obiecte.	DA
	`Image`	Afișează și manipulează fișiere grafice de pe disc sau câmpuri de tip `General` din tabele.	DA
	`Timer`	Obiect ce permite lansarea unor procese la anumite intervale de timp, fixate dinainte (spre exemplu, un text care apare/dispare, afișarea orei la fiecare secundă etc.). Invizibil la execuție.	NU
	`Page Frame`	Obiect de tip container ce permite afișarea mai multor pagini suprapuse, fiecare conținând alte controale. Foarte util pentru formularele complexe, în care spațiul de pe o singură pagină este insuficient.	NU
	`Ole Container Control`	Utilizat pentru introducerea obiectelor de tip `ActiveX`.	(în funcție de tipul obiectului)
	`Ole Bound Control`	Asemănător cu anterioru, doar că se leagă la un câmp de tip `General` dintr-o tabelă. Este utilizat în special pentru afișarea imaginilor stocate în tabelă.	DA
	`Line`	Servește la trasarea de linii pe formular.	NU
	`Shape`	Servește la trasarea de forme geometrice pe formular.	NU

Simbolul butonului	*Denumire*	*Funcţionalitate*	*Poate fi legat la o sursă de date*
	`Hyperlink`	Obiect pentru implementarea unei interfeţe tip Web (se poate combina cu componente speciale de tip `Active Doc`). Este invizibil la execuţie.	NU
	`Container`	Permite gruparea obiectelor eterogene într-un grup ce simplifică procesul manipulării lor.	NU

După cum s-a văzut şi în tabelul de mai sus, un obiect de pe un formular Visual FoxPro poate aparţine uneia dintre următoarele mari categorii:

- *container* – poate conţine alte obiecte sau containere. Se comportă ca obiect-părinte pentru alte obiecte (formularul însuşi este un obiect de tip container pentru celelalte obiecte de pe el). Toate obiectele din container moştenesc valorile setate pentru unele din proprietăţile containerului (exemplu: `Enabled`, `Visible` etc.);
- *control* – poate fi aşezat într-un container, dar nu poate fi părinte pentru alte obiecte.

Din punctul de vedere al **legăturii cu baza de date**, obiectele din formulare (controalele) pot fi clasificate astfel:

- obiecte ce se pot lega la câmpuri din tabele sau la variabile (*bound controls*) prin intermediul cărora putem manipula datele din tabele. Vom vedea mai târziu că, pentru a lega un control la o sursă de date, trebuie intervenit asupra proprietăţii `ControlSource` (`RecordSource` pentru controlul `Grid`). Datele introduse sau alese de utilizator vor fi stocate acolo unde indică proprietatea respectivă (vezi tabelul 8.2);
- obiecte ce nu pot fi legate la date (*unbound controls*), având un rol decorativ, explicativ sau de interfaţă cu alte obiecte, acestea din urmă legate ori nu la date.

Dacă nu se specifică nici o valoare validă pentru `ControlSource` (la obiectele care prezintă această proprietate), datele introduse nu sunt scrise în fişiere, nici stocate altundeva în memorie, ci rămân ca valoare a proprietăţii până la distrugerea obiectului.

Tabelul 8.2. Efectul proprietăţii `ControlSource`

Control	*Efect*
`Check Box`	În cazul în care `ControlSource` este un câmp dintr-o tabelă, atunci valoarea `NULL` arată o căsuţă inaccesibilă, valorile `.F.` sau `0` determină afişarea căsuţei „nebifate", valorile `.T.` sau `1` determină afişarea căsuţei activate („bifate"), iar valoarea `2` arată o stare incertă.
`Column` (din `Grid`)	Când `ControlSource` este un câmp dintr-o tabelă, atunci utilizatorul modifică direct valoarea acelui câmp, pe linia curentă. Deplasarea la altă linie înseamnă deplasare în tabelă pe altă înregistrare. Dacă se doreşte legarea controlului `Grid` la o întreagă tabelă, trebuie scris numele tabelei ca valoare a proprietăţii `RecordSource`.
`List Box` sau `Combo-Box`	Când `ControlSource` este o variabilă, valoarea aleasă de utilizator este stocată în variabilă. În cazul în care `ControlSource` este un câmp dintr-o tabelă, atunci valoarea aleasă este stocată în câmpul respectiv din tabelă, la întregistrarea curentă. Dacă un element este identic cu valoarea unui câmp din tabelă, atunci la deplasarea în tabelă se selectează automat alt element al listei.

Control	*Efect*
`Option Button`	Când `ControlSource` este un câmp *numeric* dintr-o tabelă, atunci în câmpul respectiv se scrie `0` (butonul neselectat) sau `1` (butonul selectat). Dacă `ControlSource` este un câmp *logic* dintr-o tabelă, atunci în câmpul respectiv se scrie `.F.` (butonul neselectat) sau `.T.` (butonul selectat). Dacă are loc deplasarea în tabelă, butoanele se activează/dezactivează automat, după valoarea curentă a câmpului corespunzător. Atenţie: în acest caz sunt *două proprietăţi* `ControlSource`: una a *grupului* şi una pentru *fiecare buton* component. Dacă `ControlSource` pentru grup este un câmp de tip şir de caractere, atunci la selecţia unui buton din grup se memorează în acel câmp textul afişat lângă butonul ales.
`Spinner`	Dacă `ControlSource` este un câmp *numeric* dintr-o tabelă, atunci valorile alese sunt scrise în câmpul respectiv, pe înregistrarea curentă.
`Text Box` sau `Edit Box`	Dacă `ControlSource` este un câmp dintr-o tabelă, atunci textele sau numerele introduse sunt scrise în câmpul respectiv, la înregistrarea curentă. Deplasarea pe altă înregistrare tabelă afectează textul/numărul afişat.

Observaţie: înainte de a lega un obiect la o variabilă independentă, trebuie avut în vedere că, la un moment dat, conţinutul său este preluat şi de proprietatea `Value`.

Principala dificultate în crearea unei interfeţe productive ţine de sincronizarea dintre starea elementelor din formular (controalele) şi conţinutul surselor de date (`ControlSource`) la care sunt legate acestea. Visual FoxPro este un mediu integrat, cu motor propriu de baze de date, dar şi cu facilităţi de programare din categoria celor „vizuale" – asemănător cu Visual Basic ori cu Delphi. Putem spune că, graţie caracterului „mai intim" al legăturii formulare-bază de date sub Visual FoxPro, modul de lucru este ceva mai uşor de înţeles.

Putem spune că, din punctul de vedere al dezvoltatorului, un formular prezintă două părţi (figura 8.3):

- partea de *front-end* sau ceea ce va vedea efectiv utilizatorul final în momentul în care va deschide formularul;
- partea de *back-end* sau mulţimea tuturor liniilor de cod introduse pentru a da funcţionalitate diverselor evenimente sau metode, precum şi mulţimea proprietăţilor configurate pentru a asigura legătura şi sincronizarea cu sursele de date.

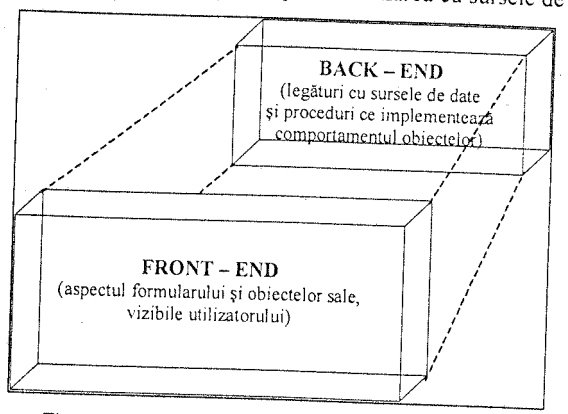


Figura 8.3. Cele două „straturi" ale unui formular

Deşi fiecare dezvoltator are stilul său propriu de lucru şi nu ne propunem să dăm aici sfaturi cu tentă general valabilă, totuşi, o *metodologie de proiectare a unui formular*, pe care o vom urma şi noi în continuare, ar *consta în următorii paşi*:

- în primul rând trebuie să fie foarte clar ce anume va face formularul respectiv, scopul acestuia şi dacă va oferi sau nu utilizatorului o interfaţă de actualizare a anumitor structuri de date;
- identificarea structurilor de date ce vor fi actualizate sau sunt necesare pentru afişare;
- crearea formularului, configurarea dimensiunilor şi adăugarea controalelor necesare. În funcţie de tipul fiecărei surse da date ce va fi actualizată, se va alege acel tip de obiect care va oferi, pe de o parte, o cât mai mare uşurinţă în utilizare, iar pe de altă parte, un risc minim de erori;
- legarea controalelor formularului la sursele de date şi, eventual, setarea valorilor altor proprietăţi, pentru „cosmetizare";
- verificarea configurărilor iniţiale ale surselor de date (deschiderea tabelelor, stabilirea indecşilor activi, valorile implicite ale variabilelor) la lansarea în execuţie a formularului;
- redactarea secvenţelor de cod aferente metodelor şi evenimentelor care definesc „comportamentul obiectelor din formular";
- testarea formularului la execuţie, corectarea erorilor găsite şi efectuarea retuşurilor finale.

O atenţie deosebită trebuie acordată alegerii celor mai potrivite tipuri de controale pentru interacţiunea cu utilizatorul, atât ca aspect, cât şi ca funcţionalitate.

Tabelul 8.3. Categorii de acţiuni care vor determina alegerea controalelor

Acţiune	*Controale recomandate*
simpla afişare a datelor	toate cele care acceptă texte şi numere (preferabil cu proprietatea `Enabled=.f.`)
oferirea unui număr de opţiuni predefinite (ex.: sex bărbătesc sau femeiesc)	`Option Group`, `List box`, `Combo-box`.
acceptarea de date ce nu pot fi prevăzute (ex.: numele şi prenumele)	`Text Box`, `Edit Box`
acceptarea de date ce trebuie să se încadreze între anumite limite (ex.: anul de studii)	`Spinner`, `Text Box` (cu validare)
oferirea posibilităţii de a declanşa anumite acţiuni predefinite (ex.: adaugă un judeţ)	`Command Button`, `Command Group`
declanşarea unei acţiuni predefinite la anumite intervale de timp (ex.: aflarea periodică a spaţiului liber de pe disc în condiţiile unor operaţii de actualizare masive)	`Timer`
un grup de obiecte trebuie să poată fi inactivat/activat (sau vizibil/invizibil), în funcţie de anumite condiţii	Obiecte plasate într-un `Container`
actualizarea unor tabele ce nu au un număr prea mare de atribute (câmpuri)	`Grid`

8.2. Macheta primului formular. Tranzacţii

În continuare vom proceda la realizarea efectivă a unui formular pentru actualizarea persoanelor din baza de date. Scopul efectiv al formularului va fi navigarea prin tabela PERSOANE, consultarea datelor corespunzătoare unor persoane şi, eventual, actualizarea tabelei: adăugarea/modificarea/ştergerea unor înregistrări. Structura tabelei PERSOANE o regăsim în capitolul 2.

În această etapă stabilim câteva detalii asupra modului în care va trebui să funcţioneze formularul în ansamblul său. Astfel, o dată lansat în execuţie, acesta trebuie să furnizeze următoarele posibilităţi:

- căutarea şi poziţionarea pe o persoană anume, pentru a vizualiza/modifica sau şterge datele referitoare la ea;
- adăugarea unei persoane noi;
- în momentul selectării unei persoane, toate controalele legate la date trebuie să afişeze datele de pe înregistrarea corespunzătoare din tabela PERSOANE;
- orice operaţie de actualizare trebuie să poată fi anulată la cererea expresă a utilizatorului.

Pentru a obţine sesiuni de actualizare provizorii, care în final se vor „comite" (permanentiza) sau nu în baza de date, în funcţie de anumite condiţii, Visual FoxPro, ca aproape orice SGBD, permite utilizarea *tranzacţiilor*. O tranzacţie (în sensul de mai sus) este un grup de instrucţiuni DML[27] înlănţuite funcţional de tipul „totul sau nimic" şi în Visual FoxPro se implementează conform următoarei scheme:

```
BEGIN TRANSACTION
Instrucţiuni de actualizare ...
END TRANSACTION
```

```
BEGIN TRANSACTION
Instrucţiuni de actualizare ...
ROLLBACK
```

Dacă tranzacţia se încheie cu `END TRANSACTION`, actualizările efectuate în baza de date de la începutul tranzacţiei (după `BEGIN TRANSACTION`) vor fi definitive; dacă tranzacţia se încheie cu `ROLLBACK`, modificările sunt abandonate, fiind restaurată situaţia existentă înainte de începutul tranzacţiei.

27. DML – *Data Manipulation Language* – instrucţiuni dedicate actualizării datelor în tabelele bazei de date. Spre exemplu, limbajul SQL conţine următoarele instrucţiuni DML: *insert*, *update*, *delete*.

```
* 1. tabela LOCALITATI va ramane intacta
BEGIN TRANSACTION
    INSERT INTO localitati VALUES('9999','Test 1','IS')
    BROWSE      && urmariti rezultatul
    BEGIN TRANSACTION
        UPDATE localitati SET jud='VS' WHERE jud='IS'
    END TRANSACTION
    BROWSE      && urmariti rezultatul
ROLLBACK
BROWSE          && urmariti rezultatul

* 2. singura modificare in tabela LOCALITATI va fi aparitia
** unei   localitati noi: "Test 1", din judetul IS
BEGIN TRANSACTION
    INSERT INTO localitati VALUES('9999','Test 1','IS')
    BROWSE      && urmariti rezultatul
    BEGIN TRANSACTION
        UPDATE localitati SET jud='VS' WHERE jud='IS'
    ROLLBACK
BROWSE          && urmariti rezultatul
END TRANSACTION
```

Figura 8.4. Două exemple de imbricare a tranzacțiilor

Există câteva ***restricții legate de tranzacții***:

- într-o tranzacție nu pot fi utilizate instrucțiuni DDL (*Data Definition Language*) – CREATE, MODIFY, CREATE TABLE, ALTER TABLE etc. – și nici comanda PACK;
- nu se poate părăsi mediul Visual FoxPro dacă există tranzacții deschise;
- tranzacțiile Visual FoxPro pot fi imbricate, dar nu mai mult de 5 niveluri;
- în cazul mai multor tranzacții imbricate (incluse), anularea unei tranzacții de pe un nivel superior atrage implicit anularea tranzacțiilor de pe nivelurile inferioare, chiar dacă acestea din urmă au fost comise (s-au încheiat prin END TRANSACTION);
- tranzacțiile reprezintă o potențială sursă de erori atunci când se lucrează în rețea, cu tabele deschise în mod partajat (*buffering*). Mecanismul buffering oferă o alternativă la tranzacții (vezi și capitolul 13).

Pentru exemplificarea modului de lucru cu mai multe tranzacții, recurgem la secvențele de cod prezentate în figura 8.4. În primul exemplu, deși a doua tranzacție s-a încheiat prin permanentizarea actualizării câmpului Jud, prima s-a încheiat cu anulare, ceea ce a determinat automat anularea și celei de-a doua. În schimb, în cel de-al doilea exemplu, tranzacția inferioară rămâne anulată chiar dacă cea superioară este comisă definitiv. Pentru aflarea nivelului unei tranzacții deschise se folosește funcția: TXNLEVEL(), care va returna numărul nivelului tranzacției curente (între 1 și 5) sau 0, dacă nu este nici o tranzacție deschisă. Comanda BEGIN TRANSACTION incrementează acest număr, END TRANSACTION sau ROLLBACK îl decrementează.

Vom reveni la formular urmând pașii pe care i-am prezentat ceva mai sus. O dată ce formularul nostru își propune să actualizeze tabela PERSOANE, este foarte clar că una dintre structurile de date implicate va fi chiar tabela în discuție (câmpurile acesteia). Pe lângă aceasta mai avem nevoie, însă, de încă două tabele. Să ne oprim puțin asupra atributului *cheie străină*, Codpost. Acest câmp ar trebui completat cu codul poștal al

localității de domiciliu a persoanei, dar, de obicei, ne este mai la îndemână numele localității și, eventual, al județului, și mai puțin codul poștal. Iată de ce interfața (partea de *front-end*) trebuie să-i ofere utilizatorului posibilitatea să aleagă o localitate anume dintr-o listă de localități, eventual cu județul asociat. În urma unei asemenea opțiuni, partea invizibilă a formularului va avea grijă să introducă în câmpul PERSOANE.Codpost valoarea codului poștal corespunzător localității alese. Cum nici numele localității, nici numele județului nu se găsesc în tabela PERSOANE, în sursele de date utilizate pentru acest formular se includ și tabelele LOCALITATI și JUDETE.

Următorul pas constă în proiectarea efectivă a formularului. Crearea unui formular (form) nou se realizează folosind una dintre metodele prezentate la începutul acestui capitol.

Observație: adăugarea controalelor se realizează prin selectarea butonului corespunzător tipului de control dorit, din bara de controale, și descrierea unui dreptunghi pe formular, în locul în care trebuie să apară controlul.

O dată trasate toate controalele de care avem nevoie, vom trece la aranjarea lor (poziționare și redimensionare) îmbinând cât mai bine aspectul cu funcționalitatea. În urma acestor acțiuni vom obține un formular care arată relativ bine, dar deocamdată nu folosește la nimic (figura 8.5).

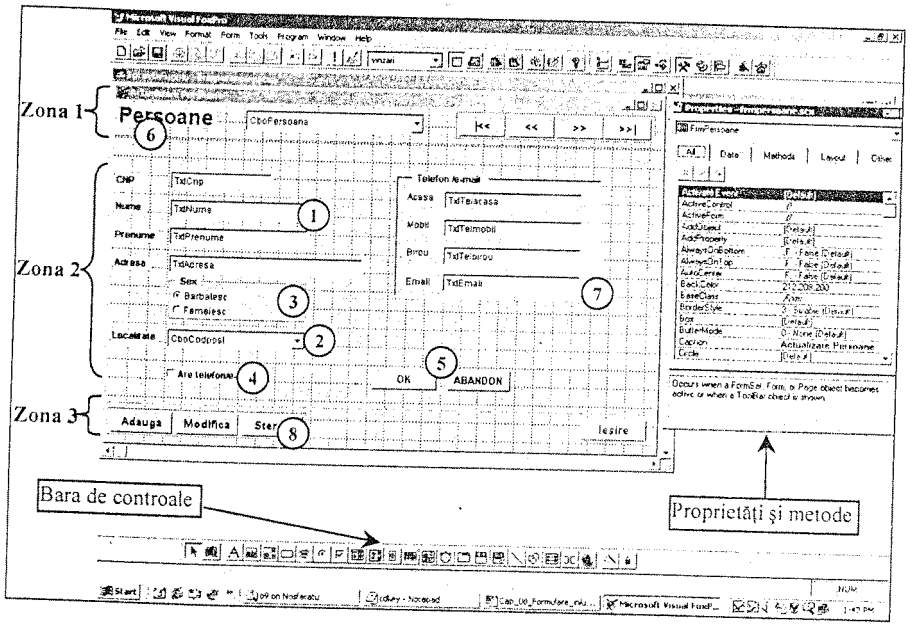


Figura 8.5. Formular pentru actualizarea tabelei PERSOANE în faza de proiectare

Pentru a obține un spor de comoditate la utilizare, un formular de acest gen se împarte convențional în trei zone distincte, astfel:

- zona 1 – conține controale ce permit navigarea în tabela (tabelele) sursă;

- zona 2 – cuprinde obiectele de afişare/editare a conţinutului surselor de date şi două butoane de validare sau anulare a ultimelor intervenţii ale utilizatorului;
- zona 3 – cuprinde butoane ce permit declanşarea unei acţiuni anume asupra surselor de date (adăugare/modificare/ştergere) şi un buton de ieşire din formular.

Atunci când utilizatorul se află într-o sesiune de modificare a înregistrării curente (zona 2), celelalte două zone sunt inactive şi invers. Astfel, o dată intrat într-o sesiune de actualizare, utilizatorul va trebui în final să o valideze sau să o anuleze şi abia apoi să aibă posibilitatea să treacă la altă înregistrare şi/sau să intre într-o nouă sesiune de modificări. Acesta este un stil de lucru recomandat pentru formularele care actualizează un număr relativ mare de surse de date (multe câmpuri ale unei/unor tabele) şi/sau în cazurile când nici un obiect de actualizare nu poate fi lăsat fără valoare introdusă.

Tabelul ce urmează îşi propune o prezentare sistematizată a tuturor obiectelor formularului din figura 8.5, în ordinea numerotării lor în figură (un număr corespunde unui anumit tip de obiect).

Tabelul 8.4. Principalele obiecte ale formularului

Tip Obiect	*Nume Obiect*	*Zona*	*Funcţionalitate*
1 **TextBox**	txtCnp	2	Afişare/editare câmp Clienti.Cnp
	txtNume	2	Afişare/editare câmp Clienti.Nume
	txtPrenume	2	Afişare/editare câmp Clienti.Prenume
	txtAdresa	2	Afişare/editare câmp Clienti.Adresa
	txtTelacasa	2	Afişare/editare câmp Clienti.Telacasa
	txtTelmobil	2	Afişare/editare câmp Clienti.Telmobil
	txtTelbirou	2	Afişare/editare câmp Clienti.Telbirou
	txtEmail	2	Afişare/editare câmp Clienti.Email
2 **Combo-Box**	cboPersoana	1	Cuprinde lista persoanelor din baza de date în ordine alfabetică. La selecţia unei persoane anume se realizează poziţionarea cursorului pe înregistrarea corespunzătoare în tabela-sursă şi actualizarea controalelor de editare din Zona 2. Nu este legat la nici o sursă de date.
	cboCodpost	2	Cuprinde lista localităţilor şi codurile poştale din tabela LOCALITATI. Este legat la câmpul CLIENTI.Codpost ca sursă de date pentru afişarea localităţii persoanei curente şi editarea acesteia. La selecţia unei localităţi se actualizează valoarea curentă a codului poştal (câmpul-sursă).
3 **Option Group**	opgSex	2	Desemnează sexul persoanei. NU este legat direct la o sursă de date. Afişarea şi, eventual, editarea sexului persoanei curente se realizează *la declanşarea anumitor evenimente* (Refresh sau InteractiveChange). Include 2 butoane: BtnB şi BtnF.
4 **CheckBox**	chkTel	2	Introdus mai mult cu titlu de exemplu. Nu este legat la nici o sursă de date. Funcţionează ca un întrerupător: – dacă este bifat, se activează containerul 7 cu obiectele de editare a telefonului/e-mailului;

Tip Obiect	*Nume Obiect*	*Zona*	*Funcţionalitate*
			– dacă nu este bifat, containerul respectiv nu este activ, utilizatorul neavând acces la obiectele acestuia.
5 **Command Button**	cmdOk	2	Salvează actualizările sesiunii curente de modificări.
	cmdAbandon	2	Anulează ultimele modificări (ultima sesiune).
	cmdIesire	3	Pentru părăsirea formularului.
7 **Container**	ctnTelefon	2	Introdus, ca şi chkTel, pentru exemplificarea modalităţii de accesare a obiectelor dintr-un alt container decât formularul însuşi. Conţine patru textbox-uri. Este activ sau inactiv, în funcţie de starea curentă a căsuţei de opţiune chkTel.
8 **Command Group**	cgrActualizare	3	Conţine trei butoane de comandă pentru adăugare (cmdAdauga), modificare (cmdModifica), ştergere (cmdSterge).
	cgrNavigare	1	Conţine patru butoane de navigare la apăsarea cărora se realizează un salt înainte sau înapoi cu o înregistrare sau un salt la prima sau ultima înregistrare.

În afara celor cuprinse în tabel, mai există obiecte de tip Label (tip notat cu 6 în figură), precum şi formularul însuşi, care are şi el un nume, frmPersoane. Obiectele Label pot avea fiecare nume adecvate, dar, dat fiind rolul lor pasiv, le putem lăsa numele implicite, dacă nu avem de gând să le editâm prin program.

Cât priveşte denumirea controalelor, aceasta trebuie să fie sugestivă (să arate pe cât posibil sursa de date a controlului sau acţiunea pe care o permite). Pentru uşurinţa înţelegerii codului-sursă, este recomandabilă aşa-numita „notaţie ungurească", în care tipul controlului este indicat de un prefix format din trei litere ataşat numelui. Aceste prefixe sunt prezentate în tabelul 8.5.

Trebuie să mai spunem aici că *în cazul containerelor, modalitatea de lucru este următoarea*:

- se adaugă containerul pe formular;
- se efectuează un click-dreapta pe acesta şi, din meniul contextual, se alege opţiunea Edit, care va schimba starea din modul selectat în mod editare;
- se adaugă celelalte obiecte necesare, prin trasare în container.

Alinierea obiectelor în formular presupune utilizarea barei cu butoanele de aliniere Layout (sau Format→Align din meniul principal), după ce în prealabil au fost selectate controalele de aliniat.

După cum se observă şi din figură, pe formularul aflat în faza de proiectare, pentru unele obiecte numele acestora este vizibil (TextBox şi Combo-Box), în timp ce pentru altele se afişează şirul de caractere introdus ca valoare a proprietăţii Caption (Command Button, Option Button, CheckBox etc.). Numele obiectelor nu vor fi vizibile la execuţia formularului, în timp ce valorile proprietăţilor Caption, da.

Tabelul 8.5. Prefixe recomandabile la denumirea controalelor

Tip control	*Prefix*
Check Box	chk
Command Button	cmd
Command Group	cgr
Container	ctn
Edit Box	edt
Form	frm
Label	lbl
Option Button	opt
Option Group	opg
Text Box	txt

Tip control	*Prefix*
Line	lin
Grid	grd
Image	img
Shape	shp
Spinner	spn
List Box	lst
Page Frame	pfr
Timer	tmr
Combo-box	cbo, cmb

Înainte de a trece la partea cea mai „sensibilă" a realizării unui formular, ce constă în legarea obiectelor la sursele de date şi codificarea comportamentului fiecăruia, trebuie să lămurim mai bine ce sunt proprietăţile şi metodele despre care am vorbit în treacăt până acum.

8.3. Proprietăţi, metode şi evenimente

Pe scurt, *proprietăţile* sunt variabile locale ale fiecărui obiect, numite şi atribute, ce pot lua diverse valori, în conformitate cu funcţionalitatea acestora. *Metodele* sunt proceduri specifice asociate fiecărui obiect în parte, ce se execută în momentul invocării lor *explicite* printr-o linie de cod dintr-o altă procedură. Metodele se pot executa şi automat, la apariţia unui *eveniment*. Evenimentele reprezintă una dintre pietrele de temelie ale sistemelor de operare de tip Windows: ele desemnează acţiuni ale utilizatorului (click-uri, deplasarea mouse-ului, apăsarea unei taste sau unei combinaţii de taste, scurgerea unui interval de timp, deschiderea unei baze de date, o cerere a sistemului de operare adresată unei aplicaţii etc.).

Prin intermediul evenimentelor şi al metodelor (se mai numesc proceduri-eveniment) se implementează comportamentul obiectului. Proprietăţile, metodele şi evenimentele unui obiect sunt întotdeauna accesibile codului-sursă implementat la nivel superior (spre exemplu, la nivel de metode ale formularului); reciproc, proprietăţile, metodele şi evenimentele formularului pot fi accesate din cadrul codului aferent unui obiect anume de pe formular.

Sintaxa generală pentru a accesa o proprietate sau o metodă a unui obiect este următoarea:

```
numecontainer.numeobiect.numeproprietate[=valoare]
```

sau:

```
variabila= numecontainer.numeobiect.numeproprietate
```

sau:

```
numecontainer.numeobiect.numemetoda
```

Containerul de nivelul cel mai înalt este formularul însuşi, care în cod se specifică prin expresia `THISFORM`. De asemenea, în codul asociat unui eveniment, numele controlului pentru care se tratează evenimentul poate fi înlocuit cu sinonimul `THIS`.

8.3.1. Despre proprietăţi

Vom prezenta în continuare o parte din cele mai importante *proprietăţi* comune mai multor obiecte şi absolut necesare în proiectarea unei interfeţe viabile. Nu vom face aici referire la cele mai puţin uzuale (pentru că s-ar putea să nu le folosiţi niciodată) sau cele privind aspectul final al obiectului (`FontColor`, `BackColor` etc.); acestea sunt mai mult decât intuitive în utilizare.

- `Value` – stochează valoarea curentă a obiectului pe parcursul execuţiei formularului. Pentru obiecte de tip listă, această proprietate preia valoarea elementului curent selectat. Pentru obiectele legate la o sursă de date, această proprietate are aceeaşi valoare ca şi valoarea curentă a sursei de date. Valorile permise pentru proprietatea `Value` sunt redate în tabelul 8.6.

Tabelul 8.6. Tipuri de date ale proprietăţii `Value`

Control	*Posibile tipuri de date pentru proprietatea* `Value`
CheckBox	Integer, Logical, Numeric
Combo-Box	Character, Integer, Numeric
CommandGroup	Character, Integer, Numeric
EditBox	Character, Memo
Grid	Character, Numeric
ListBox	Character, Integer, Numeric
OptionButton	Integer, Logical, Numeric
OptionGroup	Character, Integer, Numeric
Spinner	Currency, Integer, Numeric
TextBox	Orice tip

Notă. În cazul selecţiei unui element din grupurile de tip `OptionGroup` sau `CommandGroup`, proprietatea `Value` ia ca valoare numărul de ordine al butonului selectat.

- `ControlSource` – specifică sursa de date a unui obiect legat şi poate fi un câmp al unei tabele sau o variabilă (eventual un element al unui tablou). Valoarea curentă a sursei este afişată în controlul respectiv, iar o eventuală modificare a valorii obiectului antrenează modificarea valorii sursei de date. Este, fără îndoială, cea mai importantă proprietate a unui obiect, prin intermediul ei realizându-se legătura efectivă dintre baza de date şi elementele interfeţei-utilizator.
- `RowSource` (se întâlneşte la controalele `Combo-Box` şi `ListBox`) – specifică sursa pentru elementele ce vor popula o listă. Este diferită de `ControlSource` care, în cazul listelor, constituie, pe de o parte, sursa de afişare a unei valori (dacă ea coincide cu un element din listă), iar pe de altă parte, „ţinta" de stocare a unei eventuale modificări. Cu alte cuvinte, când alegem un element dintr-o listă, acesta va fi stocat în `ControlSource`, iar când modificăm prin program valoarea câmpului sau variabilei din `ControlSource`, obiectul va afişa un alt element din listă (dacă noua valoare coincide cu un element din listă);
- `RowSourceType` – dedicată setării tipului sursei elementelor unui obiect de tip listă. Tipul ales trebuie să fie obligatoriu compatibil cu sursa specificată în `RowSource`. Tabelul 8.7 prezintă variante posibile ale acestei proprietăţi:

Tabelul 8.7. Proprietatea RowSourceType – variante de configurare

Variantă	*Semnificație*
1 - Value	pentru valori introduse direct în RowSource (separate prin virgulă[28])
2 - Alias	specificarea câmpurilor unei tabele. RowSource va conține în acest caz numele tabelei (vor fi luate în considerare toate câmpurile)
3 - SQL Statement	specificarea unei fraze SELECT - SQL ca și RowSource. Fraza respectivă trebuie să aibă ca rezultat un cursor sau o tabelă și *nu poate avea mai mult de 256 de caractere* (exemplu: SELECT denpr,codpr FROM produse INTO CURSOR CrsProduse)
4 - Query (.qpr)	specificarea unei proceduri de interogare ce creează un cursor sau o tabelă. Opțiunea oferă o mare flexibilitate și în același timp utilitate, dat fiind faptul că de multe ori nu putem obține rezultatul dorit dintr-o singură frază SQL. Procedura de interogare trebuie să existe pe disc sub forma unui fișier cu extensia .qpr (chiar dacă aplicația este compilată într-un fișier executabil) și trebuie specificată în RowSource utilizând calea completă (exemplu: d:\aplicatie\queries\localitati_si_coduri.qpr).
5 - Array	sursa de populare a listei va fi constituită din elementele unui tablou specificat în RowSource prin nume (fără dimensiuni)
6 - Fields	lista este populată numai cu valorile câmpurilor specificate, utilizând formatul: numetabelă.câmp2,câmp1... (exemplu: Produse.denpr,codpr).
7 - Files	lista de fișiere din directorul curent, cu posibilitatea navigării prin directoare și chiar a schimbării discului curent. Pentru filtrarea fișierelor se folosește o mască de genul: *.jpg, nume.*, w???.tmp etc.
8 - Structure	numele câmpurilor tabelei specificate în RowSource (structura acesteia)

- ColumnCount – specifică numărul de coloane ce va fi afișat într-un obiect de tip listă (în cazul Combo-Box, numai la deschiderea listei) sau Grid. Dacă acest număr este mai mare decât numărul de câmpuri al sursei de date, ultimele coloane vor fi vide;
- ColumnWidths – stabilește dimensiuni pentru coloanele unui obiect de tip listă. Se exprimă în pixeli. Dacă nu sunt specificate dimensiuni, în cazul mai multor coloane, fiecare linie a listei va fi dimensionată în funcție de lungimea valorilor de pe fiecare linie a sursei de date, rezultând uneori un aspect nedorit al listei (de exemplu, pentru o listă cu trei coloane: 250, 0, 24, unde dimensiunea 0 înseamnă ascunderea coloanei respective). Această proprietate se folosește în combinație cu ColumnLines;
- BoundColumn – specifică a câta coloană (de la stânga la dreapta, numărătoarea începe cu 1) este cea legată de sursa de date (ControlSource) sau valoarea cărei coloane va fi preluată de proprietatea Value. Se folosește doar în cazul în care un obiect de tip listă are mai multe coloane. Oricum, indiferent de numărul coloanei legate, obiectul (Combo-Box sau ListBox) va afișa elementul de pe prima coloană, corespunzător liniei curente.

28. Controalele ListBox și Combo-Box pot avea mai multe coloane, de lățimi diferite, dintre care cel mult una poate fi legată la o sursă de date.

- RecordSource – proprietate pe care o regăsim numai la controalele de tip Grid și specifică sursa de date. Această sursă este bidirecțională: se afișează datele tabelei în controlul Grid și, totodată, valorile pot fi modificate direct în grid, deplasarea pe altă linie producând actualizarea lor. Fiecare coloană a controlului Grid ar trebui să fie, la rândul ei, legată prin propria proprietate ControlSource la o coloană a tabelei-sursă. În caz contrar, grid-ul afișează coloanele în ordinea în care ele se găsesc în sursa de date;
- RecordSourceType – tipul sursei de date pentru un control Grid. Poate fi una dintre variantele din tabelul 8.8.

Tabelul 8.8. Configurări posibile ale RecordSourceType

Variantă	*Semnificație*
1-Alias	sursa va fi o tabelă din baza de date (sau o tabelă derivată – *view*)
2-Prompt	la execuție se va deschide o fereastră de dialog pentru ca utilizatorul să-și aleagă singur tabela-sursă. În acest caz, proprietatea RecordSource va rămâne vidă în faza de proiectare
3-Query (.qpr)	procedură de interogare cu extensia .qpr, ce are ca rezultat o tabelă sau o tabelă temporară (vezi și RowSourceType)
4-SQL statement	frază SELECT ce are ca rezultat o tabelă permanentă sau temporară

- Name – specifică numele obiectului, folosit pentru a accesa proprietățile și metodele obiectului respectiv. Obiectele din același container nu pot avea nume identice;
- Caption – specifică textul ce va fi afișat pentru obiectul respectiv;
- Enabled – poate lua doar două valori (.T.sau .F.) și determină starea obiectului din punctul de vedere al accesibilității de către utilizator (accesibil sau inaccesibil la momentul execuției), obiectul fiind (totuși) vizibil;
- Visible – poate fi .T.sau .F., setarea având ca rezultat vizibilitatea sau invizibilitatea obiectului pentru utilizator (tot la momentul execuției);
- SelectOnEntry – poate fi .T.sau .F.; indică dacă valoarea curentă afișată de obiect va fi (sau nu) automat selectată în momentul în care acesta primește controlul (focus-ul). Proprietatea este utilă pentru a nu obliga utilizatorul să șteargă mai întâi valoarea veche și apoi să o introducă pe cea nouă, din moment ce poate scrie direct peste valoarea veche.

Notă. Am întâlnit aici sintagma „obiectul preia controlul" și termenul „focus"; noțiunile sunt cumva sinonime și se referă la obiectul activ la un moment dat. Această stare este prezentată vizual în maniere diferite, în funcție de tipul obiectului. Astfel, controalele de tip TextBox prezintă un cursor clipitor când sunt active, controalele de tip CommandButton, OptionButton și CheckBox prezintă un dreptunghi punctat care încadrează textul afișat, controlul Label nu prezintă nimic, fiindcă nu i se poate ceda controlul, la controlul Grid una din căsuțe este afișată cu altă culoare etc. În starea de „focus", butoanele pot fi „apăsate" și cu tasta Enter sau cu tasta Space, în loc de click. Schimbarea controlului activ are loc prin click pe un alt control sau cu tastele Tab (înainte), respectiv ShiftTab (înapoi).

- TabIndex – poate lua numai valori numerice pozitive, de la 1 la *n*, și indică ordinea în care va fi transferat controlul de la un obiect la altul, în cadrul

formularului, în cazul utilizării exclusive a tastaturii în acest scop (Tab, Ctrl+Tab). Inițial este ordinea în care au fost trasate controalele, dar se poate modifica (cel mai simplu prin View→Tab Order..., urmând a se face un click pe numerele afișate pe fiecare control, în noua ordine dorită);

- TabStop – dacă este setată cu valoarea .F., obiectul nu va primi niciodată controlul în cazul utilizării exclusive a tastaturii pentru deplasare;
- ButtonCount – specifică numărul de butoane disponibile într-un obiect de tip OptionGroup sau CommandGroup;
- ToolTipText – specifică textul ce va apărea ca etichetă pentru obiectul respectiv în momentul „plimbării" mouse-ului deasupra sa. Necesită setarea unei alte proprietăți a formularului și anume: ShowTips, cu valoarea .T.
- WindowType – proprietate specifică doar formularului, ce poate avea două valori: Modeless și Modal. Dacă proprietatea are valoarea Modal, este interzis accesul la alte formulare sau la meniu atât timp cât formularul se află în execuție.

Atenție! Proprietatea WindowType *este foarte importantă atunci când dintr-un formular se apelează un altul în care trebuie efectuate imperativ o serie de operații.* Astfel, dacă formularul apelat este modal, execuția formularului apelant este suspendată până la ieșirea din formularul apelat, începând cu linia de cod imediat următoare celei care l-a lansat în execuție. Dacă formularul apelat nu este modal, execuția celui apelant nu este suspendată.

Ar mai fi de punctat și câteva ***proprietăți specifice obiectelor de tip listă*** (Combo-Box și ListBox) ce servesc la manipularea elementelor listei. După cum vom vedea și pe parcursul acestui capitol, dar și în capitolul 10, cunoașterea acestora este importantă pentru sincronizarea afișării unui element sau altul în conformitate cu valorile curente ale surselor de date. Pentru exemplificare, vom presupune că avem un obiect de tip listă, cu numele cboTest, cu două linii și două coloane cu elementele: *a11, a12, a21, a22.* Iată proprietățile:

- ListCount – returnează numărul de linii din lista de elemente. În exemplul nostru, expresia x=cboTest.ListCount va avea ca rezultat x=2;
- ListItem(nrLinie) – extrage (returnează) elementul din listă de pe linia specificată prin nrLinie. Dacă lista are mai multe coloane, elementul va fi implicit extras de pe prima coloană. De exemplu, în urma execuției instrucțiunii de atribuire x=cboTest.ListItem(2), x va avea valoarea a21;
- List(nrLinie,nrCol) – returnează elementul din listă specificat prin Nrlinie, Nrcoloană. Folosit pentru listele multicoloană. De exemplu, în urma execuției instrucțiunii: x=cboTest.List(1,2), x va avea valoarea a12;
- ListIndex – desemnează indexul liniei curente sau care va deveni curentă și, prin urmare, elementul curent afișat de control. Exemplu: cboTest.ListIndex=2 va determina selectarea liniei 2 din listă, iar cboTest va afișa ca valoare curentă a21.

8.3.2. Despre evenimente și metode

Evenimentele și *metodele* folosite mai des în proiectarea formularelor ar fi următoarele:

- Init – eveniment – se declanșează în momentul în care se creează un obiect, la lansarea în execuție a unui formular.

Observație: evenimentul Init al unui container (exemplu: formularul) se declanșează *după* ce s-au inițializat toate obiectele conținute de acesta (în ordinea adăugării lor în container).

În metoda atașată evenimentului Init se pot introduce parametrii formularului prin utilizarea, obligatoriu pe prima linie, a sintaxei:

```
LPARAMETERS parametru1,parametru2...
```

În astfel de cazuri, formularul va trebui lansat cu comanda:

```
DO FORM numeformular WITH valparam1, valparam2,...
```

Pentru a împiedica crearea unui obiect, în funcție de valoarea anumitor parametri la momentul execuției, în evenimentul Init se folosește instrucțiunea: RETURN.F. În acest caz, nici evenimentul Destroy pentru respectivul obiect nu va mai avea loc.

- Destroy – eveniment – se execută în momentul în care un obiect este distrus (șters de pe ecran și descărcat din memorie, ca de exemplu la închiderea formularului).

Observație: spre deosebire de Init, în cazul containerelor (și formularelor), evenimentul Destroy la nivel de container se declanșează *înainte* de evenimentul Destroy al fiecărui obiect conținut, astfel încât pot fi apelate obiectele interne înainte de a fi distruse.

- Load – eveniment – se declanșează *înainte de crearea oricărui obiect*, deci înainte de evenimentul Init, și îl regăsim numai la nivel de formular. Și în acest caz putem utiliza RETURN .F. pentru a împiedica lansarea în execuție a formularului, dacă anumite condiții nu sunt îndeplinite, fără ca evenimentul Destroy să fie invocat;
- Unload – eveniment – specific doar formularelor; se declanșează după evenimentul Destroy;
- Release – metodă – specifică doar formularelor; se declanșează la invocare efectivă printr-o linie de cod (exemplu: Thisform.Release) și duce la *închiderea formularului* (dispariția sa de pe ecran și ștergerea din memorie). Declanșează evenimentele Destroy, apoi Unload;
- Activate – eveniment – specific formularelor și controalelor de tip PageFrame (cadre de pagină). Se declanșează la preluarea controlului, când un formular sau un cadru de pagină (din mai multe) devine activ în urma acțiunii utilizatorului sau după apariția pe ecran a controlului, dar numai ca urmare a invocării explicite prin cod;
- GotFocus – eveniment – se declanșează în momentul când un obiect primește controlul („focus"-ul) – prin click sau deplasare cu ajutorul tastaturii;
- When – eveniment – se declanșează înainte de GotFocus și este utilizat în general pentru blocarea sau nu a accesului la un obiect în funcție de anumite condiții. Dacă

acest eveniment returnează .F. (în codul său se scrie: RETURN .F.), obiectul nu va primi controlul;

- Valid – eveniment – se declanşează chiar înainte ca un obiect să piardă controlul şi este utilizat în general pentru a nu permite „părăsirea" controlului dacă anumite condiţii nu au fost îndeplinite (spre exemplu, dacă datele introduse într-un TextBox nu sunt valide ca format sau semnificaţie). Dacă evenimentul returnează .F. (RETURN .F.), utilizatorul nu poate „pleca" de pe obiectul respectiv decât după introducerea unor date corecte. Condiţiile de validare se verifică în metoda asociată, folosind şi proprietatea Value, chiar în procedura evenimentului Valid;
- LostFocus – eveniment – se declanşează când un obiect pierde controlul („focus"-ul), la click pe alt obiect ori la navigarea cu ajutorul tastaturii. Are loc *după* evenimentul Valid;
- SetFocus – metodă – controlul va fi predat obiectului specificat (exemplu: CmdAbandon.setfocus). Nu poate fi invocată din evenimentul Valid;
- Refresh – metodă – se declanşează la invocare explicită printr-o linie de cod şi generează reactualizarea valorilor afişate de obiecte (valori stocate în proprietatea Value), în funcţie de valorile curente ale surselor de date. Apelul acestei metode pentru un container (de exemplu formular : Thisform.Refresh) va duce automat la invocarea metodei pentru fiecare obiect pe care îl conţine, provocând astfel „împrospătarea" tuturor obiectelor. În cazul utilizării controalelor PageFrame, aceeaşi comandă va genera metoda Refresh *doar pentru obiectele din pagina curentă* (activă la momentul invocării metodei). La execuţia metodei Refresh, controalele legate la câmpuri din tabele vor actualiza valorile afişate, în funcţie de înregistrarea curentă;
- Requery – metodă – specifică obiectelor de tip listă, reevaluează expresia din proprietatea RowSource şi actualizează lista de elemente a obiectului. Se declanşează la invocare explicită prin cod;
- Error – eveniment – se declanşează la apariţia unei erori ca urmare a execuţiei codului ataşat unei metode a obiectului curent. Prezintă importanţă pentru tratarea erorilor ce pot apărea la execuţia unui formular;
- InteractiveChange – eveniment – se declanşează când utilizatorul modifică valoarea unui obiect din tastatură sau mouse;
- Click – se declanşează la un click de mouse sau la apăsarea tastei Enter (ori Space) pe obiectul curent.

Întrucât momentul lansării în execuţie a unui formular este extrem de important în funcţionarea interfeţei unei aplicaţii, prezentăm în figura 8.6 ordinea declanşării evenimentelor formularului. Obiectul Data Environment – şi evenimentele asociate – va fi discutat pe larg mai târziu în cadrul acestui capitol.

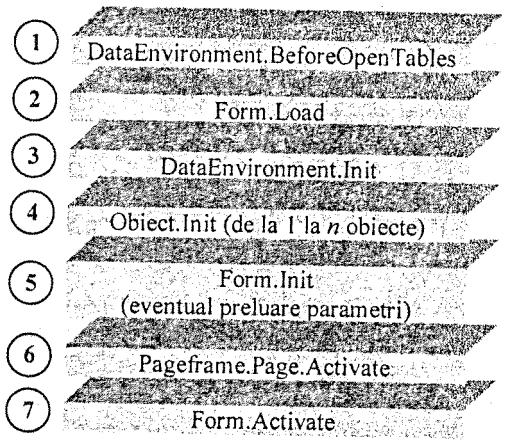


Figura 8.6. Ordinea declanşării evenimentelor dintr-un formular

În aceeaşi manieră, figura 8.7 prezintă evenimentele ce se vor declanşa la închiderea unui formular fie din butonul specific oricărei ferestre Windows, fie la invocarea metodei Release a formularului în maniera: thisform.release.

Secvenţa evenimentelor care se produc în momentul în care un obiect anume preia controlul, precum şi în momentul când acesta pierde controlul este redată în figura 8.8.

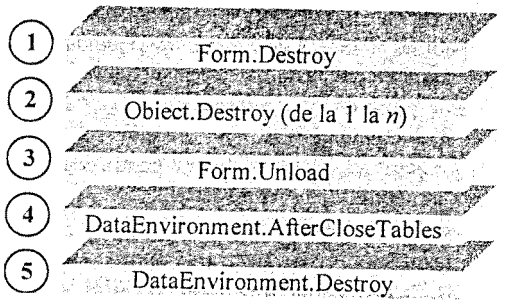


Figura 8.7. Ordinea declanşării evenimentelor în momentul închiderii unui formular

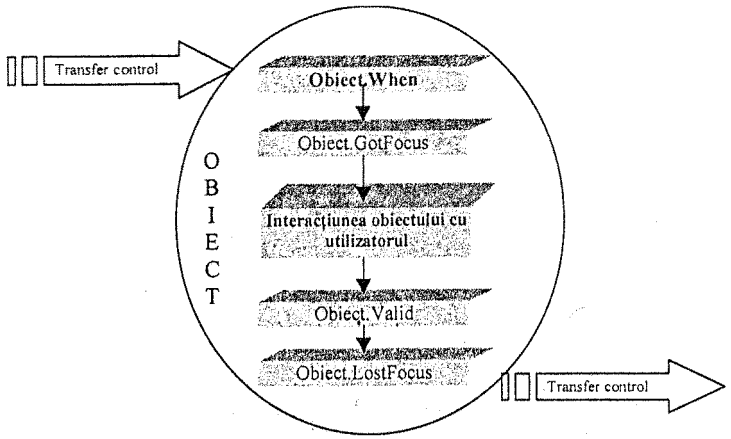


Figura 8.8. Ordinea declanşării evenimentelor când un obiect primeşte sau pierde controlul

8.4. Proprietăți ale controalelor pentru formularul în studiu

În ceea ce priveşte formularul nostru, pasul următor constă în configurarea proprietăților fiecărui control în parte. Modalitatea standard de lucru presupune următoarea secvență de acțiuni: selectăm controlul ce ne interesează şi alegem opțiunea `Properties`, fie din meniul contextual deschis cu un click-dreapta, fie din opțiunea `View→Properties` a meniului principal. Vom obține *fereastra de proprietăți şi metode pentru obiectul în cauză*, fereastră ce cuprinde, la rândul ei, cinci cadre de pagină (figura 8.9):

- `All` – lista tuturor proprietăților, evenimentelor şi metodelor obiectului selectat (evident, este cea mai lungă);
- `Data` – lista proprietăților posibil a fi setate pentru legarea obiectului la sursa de date şi formatarea modului de afişare a acesteia;
- `Methods` – metodele obiectului curent;
- `Layout` – lista proprietăților ce determină forma sub care va fi afişat obiectul (dimensiuni, culori, efecte speciale);
- `Other` – lista altor proprietăți ce nu se includ în categoriile `Data` şi `Layout`.

Valorile proprietăților se introduc în caseta arătată în figura 8.9. Unele proprietăți permit lansarea unui dialog printr-un mic buton notat cu trei puncte (...). De asemenea, o proprietate poate fi setată şi prin utilizarea unei funcții ce returnează o valoare validă. Pentru a fi evaluată, funcția trebuie obligatoriu precedată de semnul „=". Spre exemplu: `Obiect.Proprietate=IIF(condiție,ValoareDacaAdevarat,ValoareDacaFals)`. Un buton notat cu *fx*, aflat în dreapta casetei pentru scrierea valorii, permite construirea asistată a unei expresii ce conține funcții. Dacă funcția e simplă, ea se poate introduce direct, fără a omite semnul egal.

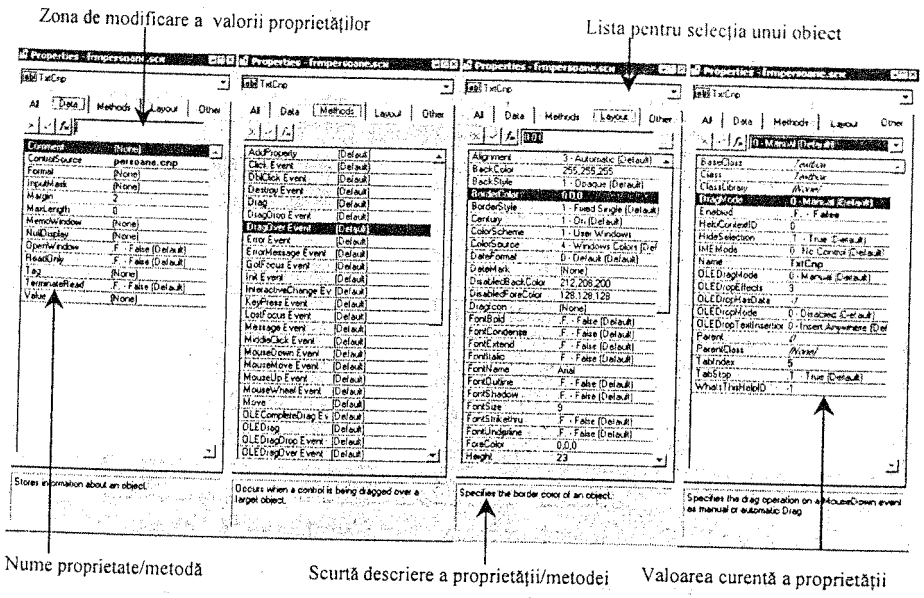


Figura 8.9. Cele patru cadre de pagină ale ferestrei `Properties`

Pentru a introduce *codul procedurilor-eveniment sau metodelor*, este necesar un click--dreapta pe obiectul dorit şi se alege din meniul contextual opțiunea `Code`, sau se poate alege, chiar din meniul principal, opțiunea `View→Code`, având selectat un obiect; şi mai simplu, se poate executa dublu-click pe obiectul pentru care se intenționează scrierea metodelor. Va apărea o fereastră în care se va scrie codul (figura 8.10).

Deplasarea la codul altei metode pentru un obiect dat poate avea loc prin selecția metodei adecvate din combo-box-ul situat în dreapta-sus (figura 8.10).

Atenție! Metodele pentru care am scris deja cod nu respectă ordinea alfabetică inițială, ci sunt plasate *la începutul listei*, cu caractere **bold**.

Accesul rapid la o anumită metodă a unui obiect are loc direct din fereastra `Properties`, după selectarea unui control şi efectuarea unui dublu-click pe numele evenimentului/metodei. În această fereastră, metodele la care s-a adăugat cod sunt evidențiate prin textul **[User Procedure]**. Tot de aici se poate şterge complet codul unei metode, prin click-dreapta pe numele metodei, urmată de alegerea opțiunii `Reset to default` din meniul contextual (a se utiliza cu precauție).

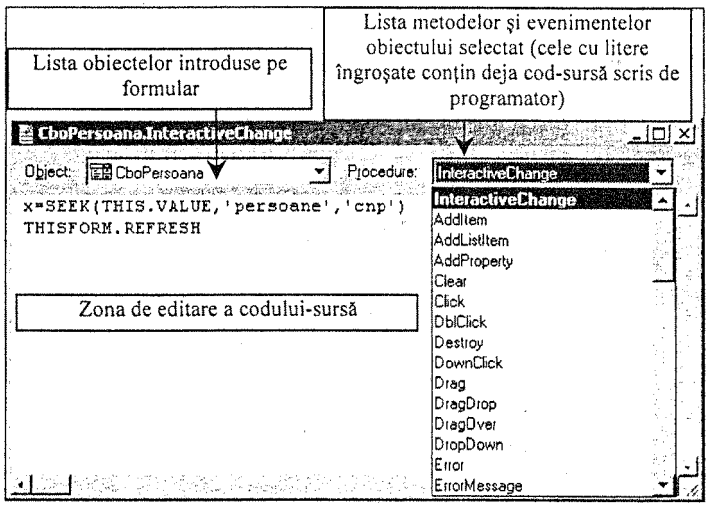


Figura 8.10. Fereastra pentru scrierea codului (se observă lista de metode)

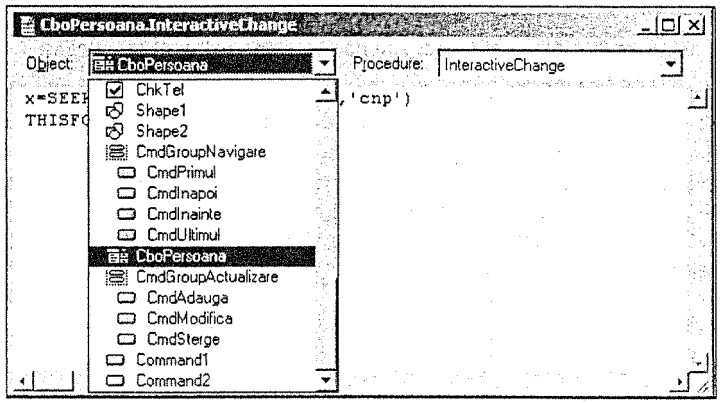


Figura 8.11. Fereastra pentru scrierea codului (se observă lista de obiecte)

În finalul acestei părţi se cuvine a face o ultimă precizare referitoare la faptul că *putem adăuga unui formular (dar numai lui, nu şi controalelor componente) proprietăţi şi metode noi (definite de utilizator)*, în funcţie de necesităţi. Vom recurge la această posibilitate pentru modularea şi eficientizarea codului. Într-adevăr, imaginaţi-vă că putem declanşa o operaţie (ce necesită 50 de linii de cod) atât prin click pe un buton, cât şi prin apăsarea unei combinaţii de taste când ne aflăm într-o căsuţă de text. Sunt cel puţin două posibilităţi:

- una complicată: scriem cele 50 de linii de cod o dată pentru evenimentul `Click` al controlului `CommandButton` şi încă o dată pentru evenimentul `KeyPress` al controlului `TextBox`;
- una simplă: la nivelul formularului declarăm o nouă metodă, să zicem `OperatieComplexa`, ce va cuprinde cele 50 de linii de cod, iar în evenimentul

`Click` al controlului `CommandButton`, respectiv în evenimentul `KeyPress` al controlului `TextBox` invocăm explicit noua metodă: `thisform.OperatieComplexa`.

Adăugarea unei noi metode are loc prin intermediul opţiunii `Form→New Method...` a meniului sistem care afişează o fereastră de dialog (figura 8.12).

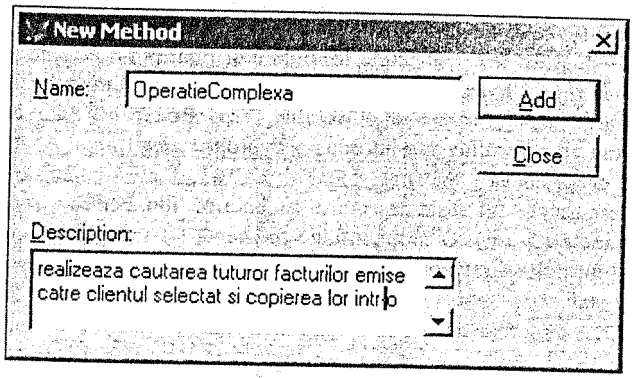


Figura 8.12. Declararea unei noi metode la nivel de formular

O proprietate a unui control poate fi apelată din alte controale numai prin **referinţa completă**: `THISFORM.numecontrol.numeproprietate` sau `THISFORM.numecontainer.numecontrol.numeproprietate` (dacă obiectul se află într-un container). Regula este valabilă şi pentru metode.

O variabilă oarecare poate fi apelată (citită/scrisă) din evenimentele sau metodele mai multor obiecte numai dacă este declarată ca fiind globală (`PUBLIC`). În general, aceste variabile se declară în evenimentul `Load` al formularului, pentru a fi accesibile tuturor metodelor „de după". Declararea poate avea loc şi într-o procedură oarecare, care va apela formularul respectiv. Mai elegant, se poate declara o proprietate nouă a formularului, ce poate fi apelată din orice punct al acestuia cu `THISFORM.numeproprietate`. Declararea unei noi proprietăţi are loc cu ajutorul meniului `Form→New Property...`, care afişează dialogul din figura 8.13.

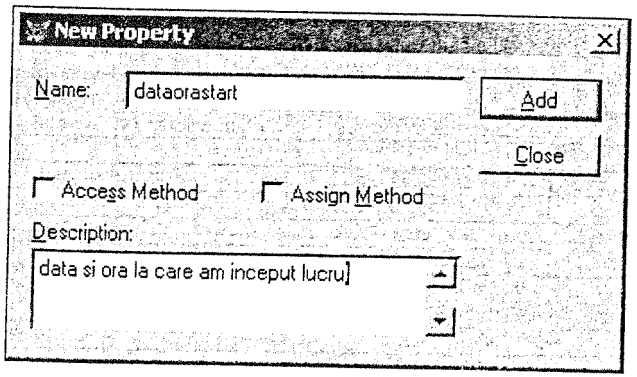


Figura 8.13. Adăugarea unei noi proprietăţi pentru un formular

Notă. Proprietăţile şi metodele adăugate de utilizator nu respectă ordinea alfabetică iniţială, fiind întotdeauna adăugate *la sfârşitul listei de proprietăţi şi metode* din fereastra Properties.

De fapt, prin adăugarea de noi proprietăţi şi metode, am făcut deja puţină programare obiectuală (asemenea unui personaj literar care făcea proză fără să ştie...). Tehnica se numeşte *subclasare* şi cu ea ne vom întâlni din nou în capitolul 11.

În tabelul 8.9 regăsim toate obiectele formularului (mai puţin cele de tip Label) cu proprietăţile setate în funcţie de necesităţi. Nu vom insista, după cum spuneam şi la început, asupra proprietăţilor de formatare a obiectului (FontBold, ForeColor, Height, Width etc.). Numele obiectului este introdus prin proprietatea Name. Atenţie la obiectele care dispun de proprietatea Caption (Form, Label, Option Button, Command Button etc.) care este textul afişat pe control (în bara de titlu, pentru Form) şi nu trebuie confundat cu numele (Name). O altă confuzie frecventă (la începători) este aceea dintre numele intern al formularului (proprietatea Name) şi numele fişierului (cu extensia .scx) în care este salvat – numele fişierului este cel care apare în secţiunea Documents→Forms a ferestrei Project Manager.

Tabelul 8.9. Precizări privind controalele formularului frmPersoane

Proprietate	*Valoare*	*Comentarii*
Control: txtCnp		
ControlSource	PERSOANE.Cnp	Controlul este legat la atributul CNP din tabela PERSOANE.
Enabled	.F. - False	La lansarea formularului, controlul este dezactivat (pentru lămuriri, vezi împărţirea pe zone a formularului prezentată la începutul capitolului).
SelectOnEntry	.T. - True	La preluarea controlului, în momentul execuţiei, conţinutul obiectului va fi implicit selectat. Setare foarte utilă dacă nu dorim ca utilizatorul să fie nevoit să şteargă mai întâi conţinutul vechi şi apoi să introducă noua valoare.
TabIndex	5	Numărul de ordine al obiectului, ce va determina la al câtelea TAB (sau Enter, sau tastă cu săgeată) acesta va primi controlul în cazul utilizării exclusiv a tastaturii pentru navigare (este ignorat în cazul utilizării mouse-ului).
Control: txtNume		
ControlSource	PERSOANE.Nume	
Enabled	.F. - False	
SelectOnEntry	.T. - True	
TabIndex	6	
Control: txtPrenume		
ControlSource	PERSOANE.Prenume	
Enabled	.F. - False	
SelectOnEntry	.T. - True	
TabIndex	7	

Proprietate	*Valoare*	*Comentarii*
Control: txtAdresa		
ControlSource	Persoane.Adresa	
Enabled	.F. - False	
SelectOnEntry	.T. - True	
TabIndex	8	
Control: txtTelacasa		
ControlSource	PERSOANE.Telacasa	
Enabled	.T. - True	Este activ în momentul lansării în execuţie a formularului, dar containerul în care se află va fi inactiv. Ca urmare, şi obiectul va fi inactiv.
SelectOnEntry	.T. - True	
TabIndex	21	
Control: ctnTelacasa (în containerul ctnTelefon)		
Enabled	.F. - False	
TabIndex	12	
SpecialEfect	1-Sunken	Aspect 3D de „scufundare".
Control: txtTelmobil (în containerul ctnTelefon)		
ControlSource	PERSOANE.Telmobil	
Enabled	.T. - True	Vezi txtTelacasa.
SelectOnEntry	.T. - True	
TabIndex	22	
Control: txtTelbirou (în containerul ctnTelefon)		
ControlSource	PERSOANE.Telbirou	
Enabled	.T. - True	Vezi txtTelacasa.
SelectOnEntry	.T. - True	
TabIndex	23	
Control: txtEmail (în containerul ctnTelefon)		
ControlSource	PERSOANE.Email	
Enabled	.T. - True	Vezi txtTelacasa.
SelectOnEntry	.T. - True	
TabIndex	24	
Control: opgSex		
ButtonCount	2	Include două butoane de opţiune.
Enabled	.F. - False	
TabIndex	9	
Control: BtnB (în grupul opgSex)		
Caption	Barbatesc	Textul ce apare lângă buton.
Name	BtnB	Numele obiectului.
Value	1	Este butonul implicit selectat la lansarea formularului.
Control: BtnF (în grupul opgSex)		
Caption	Femeiesc	
Name	BtnF	
Value	0	Este deselectat la lansarea formularului. Folosim această proprietate pentru a stabili, din faza de proiectare, care opţiune este implicit selectată din cele existente într-un OptionGoup.

Proprietate	*Valoare*	*Comentarii*
Control: chkTel		
Caption	Are telefon/email	Text ce apare în dreapta căsuței de bifat.
Enabled	.F. - False	
Style	0	0 - Standard înseamnă căsuță de opțiune „obișnuită". 1 - Graphical conferă aspectul tridimensional (asemănător unui buton de comandă), apăsat când opțiunea este activată.
TabIndex	11	
Control: cboCodpost		
ControlSource	PERSOANE.Codpost	Specifică atributul de care este legată lista și a cărui valoare curentă va constitui: – sursa de afișare a unui element anume din listă, corespunzător valorii curente a câmpului; – „ținta" de modificare a valorii prin alegerea unui alt element din listă. Proprietatea DataSource va fi obligatoriu un câmp de tip șir de caractere. Acest tip de control nu se poate lega la o sursă de date de tip numeric.
Enabled	.F. - False	
TabIndex	10	
RowSourceType	3-SQL Statement	Sursa de populare a listei va fi rezultatul unei fraze SELECT SQL.
RowSource	SELECT loc, judet,codpost FROM localitati l, judete j WHERE l.jud=j.jud ORDER BY loc INTO CURSOR crslocalitati	Fraza SELECT ce furnizează elementele listei. Când lista nu este deschisă, obiectul va afișa elementul curent al primei coloane. Observăm că lista este ordonată după numele localității, pentru a furniza utilizatorului posibilitatea căutării rapide (vezi proprietatea Style).
BoundColumn	3	A câta coloană din cele furnizate de RowSource este cea legată la Control Source. În cazul nostru, este coloana pe care avem codul poștal, întrucât lista servește la completarea câmpului PERSOANE.Codpost, pentru o persoană nou-introdusă sau care-și schimbă domiciliul.
ColumnCount	2	Numărul de coloane afișate la deschiderea listei (click, sau Space, sau Alt+↓ pe control), numărându-le de la stânga la dreapta pe cele enumerate în RowSource. Dacă dorim să fie vizibil și codul poștal, vom seta proprietatea pe valoarea 3.
ColumnWidths	200, 150	Lățimea fixă a coloanelor ce vor fi afișate pentru aspectul estetic al listei. Dacă dorim să ascundem județul, vom introduce valoarea 3 în ColumnCount și 200, 0, 50 pentru această proprietate.

Proprietate	*Valoare*	*Comentarii*
Column Lines	.T.	Linii care separă coloanele în partea „desfășurată" a listei.
Style	2-DropDownList	Proprietate extrem de importantă. Poate avea două valori: 0 - DropDownCombo – utilizatorul va avea posibilitatea fie să aleagă un element din listă, fie să-l introducă de la tastatură (pentru a-l adăuga în listă, trebuie scris cod în evenimentul Valid); deși utilă în anumite cazuri, această setare este o sursă probabilă de erori; 2 - DropDownList – nu va exista decât posibilitatea alegerii unui element din listă. În cazul în care lista este ordonată, se poate căuta rapid în listă prin apăsarea primelor 2-3 caractere ale elementului dorit (proprietatea IncrementalSearch trebuie să aibă valoarea .T.).
Control: cboPersoana		
ControlSource	(None)	Are doar rol de poziționare pe o persoană și de „împrospătare" a controalelor ce afișează conținutul înregistrării curente. Chiar dacă lista nu este legată la o sursă de date, proprietatea Value va stoca (la execuție) valoarea curentă de pe coloana specificată în proprietatea BoundColumn.
Enabled	.T.-True	
TabIndex	1	
RowSourceType	3-SQL Statement	Vezi cboCodpost.
RowSource	SELECT ALLTRIM(nume)+" "+ ALLTRIM(prenume) AS nume,cnp FROM PERSOANE WHERE NOT DELETED() ORDER BY nume,prenume INTO CURSOR crspersoane	Prima coloană a interogării SELECT este compusă (nume + spațiu + prenume), pentru că obiectul afișează în mod curent (când lista nu e deschisă) doar elementul de pe prima coloană. Nu vor fi luate în considerare înregistrările marcate pentru ștergere din tabela PERSOANE („WHERE NOT DELETED( )").
BoundColumn	2	Vezi cboPersoana.ControlSource.
ColumnCount	2	Vezi cboCodpost.
ColumnLines	.T.	
ColumnWidths	200,200	Vezi cboCodpost.
Style	2-DropDownList	Vezi cboCodpost.

Proprietate	*Valoare*	*Comentarii*
Control: cgrNavigare		
ButtonCount	4	Include patru butoane de comandă cu numele: cmdPrimul, cmdInapoi, cmdInainte, cmdUltimul.
Enabled	.T.-True	
TabIndex	2	
Control: cgrActualizare		
ButtonCount	3	Include trei butoane de comandă: cmdAdaugă, cmdModifică, cmdSterge.
Enabled	.T.	
TabIndex	4	
Controale: cmdOK şi cmdAbandon		
Caption	OK, respectiv Abandon	
Enabled	.T.	
TabIndex	13, respectiv 14	
Obiect: frmPersoane (formularul însuşi)		
Caption	Actualizare Persoane	Text ce apare ca titlu al ferestrei formularului.
WindowState	1-Modal	Utilizatorul nu va avea acces la meniul aplicaţiei (sau la alt formular) decât după ieşirea din acest formular.
Asters	.F. - False	Proprietate adăugată de utilizator (ca în figura 8.13) ce ia valoarea .T. dacă utilizatorul a şters măcar o persoană. Este folosită pentru lansarea comenzii PACK (ştergere definitivă) la ieşirea din formular, dar numai dacă în sesiunea curentă s-a marcat pentru ştergere măcar o înregistrare. Este necesară deoarece comanda PACK consumă un timp destul de mare la execuţie pentru tabelele cu multe înregistrări, mai ales dacă au şi indecşi.

Precizare. În cazul containerelor (Container), pentru grupurile de butoane (CommandGroup sau OptionGroup), manipularea butoanelor componente (de exemplu, în scopul rearanjării lor) presupune utilizarea opţiunii Edit din meniul contextual, ce se deschide la un click-dreapta pe containerul-grup (el apare încadrat cu linie groasă, haşurată după această operaţie).

8.5. Sursele de date ale controalelor

O dată ajunşi în această fază a realizării formularului, urmează să parcurgem următorul pas, prin care ***ne vom asigura că toate sursele de date vor fi deschise/iniţializate*** în momentul lansării formularului în execuţie, când se creează efectiv obiectele legate de ele.

Este foarte important acest aspect, pentru că, după cum ştim (vezi capitolul 3), înregistrările unei tabele sunt accesibile numai dacă aceasta este deschisă[29].

Avem două posibilităţi de rezolvare a problemei:

- fie utilizăm sursa de date a formularului – Data Environment – ce se iniţializează după cum am văzut şi în figura 8.6, înainte de iniţializarea obiectelor form-ului;
- fie introducem câteva instrucţiuni de deschidere a tabelelor şi, eventual de iniţializare a unor variabile globale, în evenimentul Load al formularului care de asemenea se declanşează la înainte de iniţializarea celorlalte obiecte (figura 8.6.)

8.5.1. Utilizarea Data Environment pentru deschiderea tabelelor

Data Environment (vom folosi traducerea „sursă de date", deşi literal ar fi „mediu de date") este un obiect specific formularelor şi rapoartelor Visual FoxPro, care permite selecţia tabelelor şi câmpurilor folosite în formularul sau raportul respectiv, şi memorarea tuturor referinţelor necesare în acelaşi fişier ce conţine codul formularului/raportului.

După cum se observă şi în figura 8.14, din meniul contextual deschis cu un click-dreapta pe formular se alege opţiunea Data Environment; apoi, din noul meniu contextual se alege mai întâi opţiunea Add, care va afişa o fereastră ce conţine tabelele bazei de date. În cazul formularului frmPersoane vom alege, bineînţeles, tabela PERSOANE.

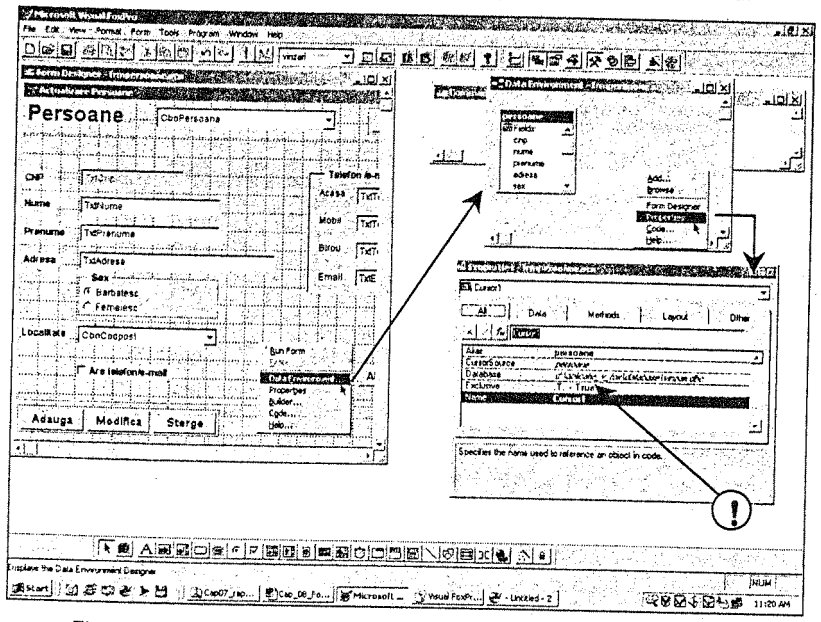


Figura 8.14. Adăugarea tabelelor necesare controalelor formularului în Data Environment

29. Constituie excepţie de la acestă regulă instrucţiunile SELECT – SQL, care deschid implicit tabela dacă nu este deschisă deja.

Observaţie. Nu adăugăm sursei de date şi celelalte tabele care am stabilit că sunt necesare formularului (`LOCALITATI` şi `JUDETE`), deoarece acestea sunt invocate în fraza `SQL` care serveşte drept `RowSource` pentru obiectul `cboCodpost` şi se vor deschide automat (dacă nu sunt deja deschise) la iniţializarea obiectelor respective. Important este ca la lansarea formularului să fie deschisă baza de date. Acest lucru se poate face printr-un program extern, executat prealabil lansării formularului, sau prin adăugarea codului pentru deschiderea bazei de date în procedura evenimentului `BeforeOpenTables` a obiectului `Data Environment` (vezi figura 8.15).

O dată adăugate tabelele necesare în `Data Environment`, este important de stabilit dacă acestea trebuie deschise *exclusiv* sau în mod *partajat* (*shared*). Setarea implicită a proprietăţii `Exclusive` (marcată cu un semn de exclamare în figura 8.14) a tabelelor din `Data Environment` este `.F.-False`. Ea nu trebuie schimbată dacă lucrăm într-un mediu multiutilizator, deoarece nimeni n-ar mai putea folosi tabelele respective până ce nu se termină lucrul cu acest formular. Dacă se lucrează doar în regim monoutilizator (la o firmă mică, având un singur calculator), putem folosi setarea `Exclusive=.T.`, mai ales dacă se doreşte ca, la ieşirea din formular, să elimine definitiv toate înregistrările marcate pentru ştergere în sesiunea de lucru curentă (comanda `PACK` cere mod de lucru exclusiv).

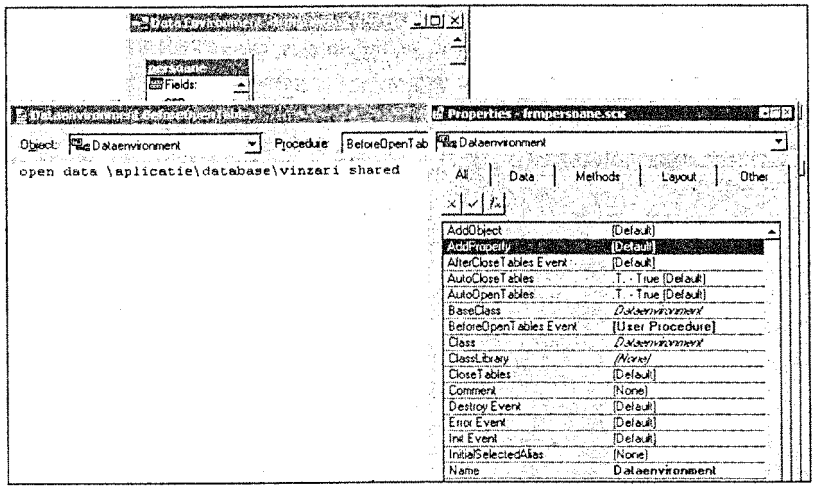


Figura 8.15. Metoda `BeforeOpenTables` a obiectului `Data Environment`

Ce facem cu `PACK` dacă suntem constrânşi să folosim modul de lucru partajat (cazul cel mai probabil în practică)? O soluţie este să renunţăm la comanda `PACK` din formular, urmând să concepem o procedură adecvată care:

- încearcă să deschidă tabelele în mod exclusiv;
- execută comanda `PACK` pe tabelele unde deschiderea exclusivă a reuşit.

O asemenea procedură poate fi găsită în capitolul 17.

8.5.2. Deschiderea explicită a bazei de date şi tabelelor

`Data Environment` este un instrument puternic, atât prin proprietăţile sale `AutoOpenTables` şi `AutoCloseTables`, cât şi prin evenimentele `BeforeOpenTables` şi `AfterCloseTables`. Se oferă astfel o cale convenabilă de a realiza rapid un formular cu un efort mai mic din partea programatorului. Cu toate acestea, `Data Environment` are şi limitări, dintre care principala este „legarea" unui formular de o anumită structură de date. Dacă utilizatorul preferă o manieră de lucru mai „migăloasă" dar mai flexibilă, el poate opta pentru deschiderea/închiderea explicită a bazei de date şi tabelelor, folosind în acest scop evenimentele formularului: `Load` pentru scrierea codului de deschidere şi `Unload` pentru închidere (dacă nu se foloseşte `Data Environment`, atunci `Load` este primul eveniment, iar `Unload` ultimul).

În cazul formularului tratat aici, codul plasat în procedura evenimentului `Load` ar putea avea următoarea formă:

Listing 8.1. Secvenţă de cod posibilă pentru evenimentul `Load`

```
if not dbused("vinzari")
    open database "\aplicatie\database\vinzari" shared
endif
if not used("persoane")
    use persoane in 0 shared
endif
```

În exemplul anterior, `\aplicatie\database\vinzari` este o *cale relativă*. Pentru a înţelege acest concept, vom preciza că elementele proiectului sunt plasate în subdirectoare distincte, dar toate în cadrul aceluiaşi director (vezi figura 8.16).

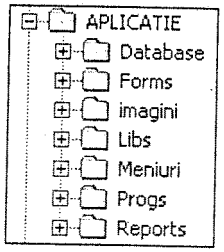


Figura 8.16. Structura de directoare a aplicaţiei

Formularul proiectat este memorat în subdirectorul `Forms`, iar fişierele bazei de date se găsesc în subdirectorul `Database`; directorul-părinte `APLICATIE` are totodată cel mai înalt nivel. Scriind calea relativă, simulăm deplasarea din directorul `Forms` la directorul rădăcină (`\`), apoi succesiv prin directoarele `APLICATIE` şi `Database`, până la fişierul `vinzari.dbc`, containerul bazei de date. Ghilimelele au rolul de a considera ca o singură expresie o cale de directoare ale căror nume conţin spaţii (de exemplu, `C:\Program Files`).

8.6. Metodele formularului şi lansarea în execuţie

Suntem acum în măsură să parcurgem şi ultimul pas, cel mai „delicat" de altminteri, ce constă în *codificarea comportamentului obiectelor*. Am văzut mai sus cum se ajunge la fereastra pentru codul-sursă aparţinând unei metode ori eveniment anume.

Vom prezenta aici codul metodelor şi evenimentelor formularului, indicând pentru fiecare secvenţă de cod în care dintre proceduri trebuie introdusă. În evenimentul `Load` al formularului vom deschide sursele de date (în cazul în care nu utilizăm `Data Environment`) şi vom iniţializa eventualele variabile globale.

După cum se poate observa şi în figura 8.17, `Activare, Dezactivare` şi `Sincronizare` (ca şi proprietatea `Asters`) sunt metode adăugate de utilizator. Am explicat anterior la ce foloseşte proprietatea `Asters`. Cele două metode, `Activare` şi `Dezactivare`, servesc la activarea/dezactivarea unor controale de editare (care trebuie să fie inactive la consultarea tabelei, pentru a nu modifica din greşeală ceva, dar, în schimb, trebuie activate la operaţiile de actualizare, pentru a putea adăuga ori modifica datele unei persoane). Cât despre cea de-a treia metodă nou-introdusă, `Sincronizare`, ea serveşte la sincronizarea listei `CboPersoana` cu modificările făcute de utilizator în tabelă (după cum ştim, obiectul nu este legat la nici o sursă de date). Deoarece ar fi ineficient să scriem aceleaşi instrucţiuni de mai multe ori (o dată pentru butonul de adăugare, o dată pentru cel de modificare etc.), este de preferat să le grupăm într-o metodă nouă, ce va fi apelată din orice punct al formularului printr-o singură linie de cod: `THISFORM.Activare` sau `THISFORM.Dezactivare`.

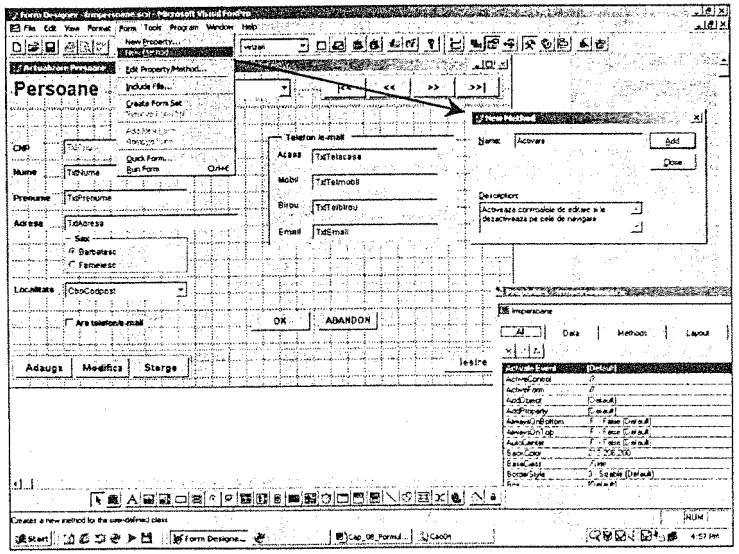


Figura 8.17. Adăugarea unei metode noi: `Activare`

Comentariile pe marginea codului sunt notate cu `&&`.

Listing 8.2. Metodele formularului `FrmPersoane`

```
                    Obiect: frmPersoane
*Eveniment: Init
THISFORM.cboPersoana.LISTINDEX=1          && obiectul va afişa implicit primul element din listă
THISFORM.cboPersoana.INTERACTIVECHANGE  && asigură "refresh-ul" controalelor legate

*Eveniment: Destroy
IF TXNLEVEL()>0                                  && daca nu e inchisa tranzactia curenta
    IF MESSAGEBOX('Salvati ultimele modificari?',32+4,'Ce facem cu ultimele modificari ?' )=6
        END TRANSACTION
    ELSE
        ROLLBACK
    ENDIF
ENDIF
IF THISFORM.Asters=.T.                           && daca a sters macar o persoana
                                                 && nota: e nevoie de acces exclusiv!!!
    SELECT persoane
    PACK
ENDIF

*Metoda: Activare
WITH THISFORM
    .txtCnp.ENABLED=.T.
    .txtNume.ENABLED=.T.
    .txtPrenume.ENABLED=.T.
    .txtAdresa.ENABLED=.T.
    .opgSex.ENABLED=.T.
    .chkTel.ENABLED=.T.
    .cmdOk.ENABLED=.T.
    .cmdAbandon.ENABLED=.T.
    .cboCodpost.ENABLED=.T.
    .cboPersoana.ENABLED=.F.
    .cgrNavigare.ENABLED=.F.
    .cgrActualizare.ENABLED=.F.
ENDWITH
THISFORM.chktel.INTERACTIVECHANGE

*Metoda: Dezactivare
WITH THISFORM
    .txtCnp.ENABLED=.F.
    .txtNume.ENABLED=.F.
    .txtPrenume.ENABLED=.F.
    .txtAdresa.ENABLED=.F.
    .opgSex.ENABLED=.F.
    .chkTel.ENABLED=.F.
    .ctnTelefon.ENABLED=.F.
    .cmdOk.ENABLED=.F.
    .cmdAbandon.ENABLED=.F.
    .cboCodpost.ENABLED=.F.
    .cboPersoana.ENABLED=.T.
    .cgrNavigare.ENABLED=.T.
    .cgrActualizare.ENABLED=.T.
ENDWITH
*Metoda Sincronizare
FOR i=1 TO THISFORM.cboPersoana.LISTCOUNT
    IF THISFORM.cboPersoana.LIST( i, 2 )==CrsPersoane.Cnp
        THISFORM.cboPersoana.LISTINDEX=i
    ENDIF
ENDFOR
```

```
THISFORM.cboPersoana.INTERACTIVECHANGE
```

Obiect: cboPersoane

```
*Eveniment: InteractiveChange
x=SEEK(THIS.VALUE,'persoane','cnp') && caută cnp-ul persoanei selectate şi se poziţionează pe
                                    && înregistrarea respectivă
THISFORM.REFRESH                    && asigură "reîmprospătarea" controalelor legate conform
                                    && înregistrării curente
```

Obiect: txtCnp

```
*Eveniment: Valid
IF LEN(ALLTRIM(THIS.VALUE))<13
    MESSAGEBOX('Codul numeric personal are obligatoriu 13 cifre',16,'CNP incorect')
    RETURN .F.                      && blochează transferul controlului la un alt obiect
ENDIF
```

Obiect: opgSex

```
*Eveniment: InteractiveChange
IF THIS.btnB.VALUE=1                && a fost selectată opţiunea 'Barbatesc'
    REPLACE PERSOANE.SEX WITH 'B'
ELSE                                && a fost selectată opţiunea 'Femeiesc'
    REPLACE PERSOANE.SEX WITH 'F'
ENDIF

*Metoda: Refresh  && nu uităm că este invocată implicit la:  thisform.refresh
IF persoane.sex='B'
    THIS.btnB.VALUE=1
    THIS.btnF.VALUE=0
ELSE
    THIS.btnB.VALUE=0
    THIS.btnF.VALUE=1
ENDIF
```

Obiect: chkTel

```
*Eveniment: InteractiveChange
IF THIS.VALUE=1                     && controlul este "bifat"
    THISFORM.ctnTelefon.ENABLED=.T.
ELSE                                && controlul nu este "bifat"
    THISFORM.ctnTelefon.ENABLED=.F.
ENDIF

*Metoda: Refresh
IF !EMPTY(persoane.telmobil) OR !EMPTY(persoane.telbirou);
      OR !EMPTY(persoane.telacasa) OR !EMPTY(persoane.email)
    THIS.VALUE=1
ELSE                                && toate cele patru câmpuri sunt vide
    THIS.VALUE=0
ENDIF
```

Obiect: cgrNavigare.cmdPrimul

```
*Eveniment: Click
SELECT CrsPersoane                  && cursorul din cboPersoane.RowSource
GO TOP
THISFORM.cboPersoana.LISTINDEX=1
THISFORM. cboPersoana.INTERACTIVECHANGE
```


Obiect: cgrNavigare.cmdUltimul

```
*Eveniment: Click
SELECT CrsPersoane
GO BOTTOM
THISFORM. cboPersoana.LISTINDEX=THISFORM.cboPersoana.LISTCOUNT
* ListCount specifică numărul total de elemente din listă
THISFORM. cboPersoana.INTERACTIVECHANGE
```

Obiect: cgrNavigare.cmdInainte

```
*Eveniment: Click
SELECT CrsPersoane
SKIP 1
IF EOF()
    MESSAGEBOX('Ati ajuns la ultima persoana', 64, ' Atentie! ' )
    GO BOTTOM
ENDIF
THISFORM.Sincronizare
```

Obiect: cgrNavigare.cmdInapoi

```
*Eveniment: Click
SELECT CrsPersoane
SKIP -1
IF BOF()
    MESSAGEBOX('Ati ajuns la prima persoana', 64,' Atentie! ' )
    GO TOP
ENDIF

THISFORM.Sincronizare
```

Obiect: cgrActualizare.cmdAdauga

```
*Eveniment: Click
SELECT MAX(codpost) FROM localitati INTO ARRAY alocdefault
BEGIN TRANSACTION
INSERT INTO persoane VALUES ('', '', '', '', 'B', alocdefault(1), '', '', '', '')
THISFORM.REFRESH
THISFORM.activare
THISFORM.txtcnp.SETFOCUS
```

Obiect: cgrActualizare.cmdModifica

```
*Eveniment: Click
BEGIN TRANSACTION
THISFORM.activare
THISFORM.txtcnp.SETFOCUS
```

Obiect: cgrActualizare.cmdSterge

```
*Eveniment: Click
SELECT persoane
IF MESSAGEBOX(' Sunteti sigur? ', 32+4, ' Persoana curenta va fi stearsa !!!' )=6
    DELETE
    THISFORM.asters=.T.
ENDIF
```

Obiect: cmdOk

```
*Eveniment: Click
END TRANSACTION
THISFORM.dezactivare
THISFORM. cboPersoana.REQUERY
THISFORM.Sincronizare
```

```
                    Obiect: cmdAbandon
*Eveniment: Click
ROLLBACK
THISFORM.dezactivare

                    Obiect: cmdIesire
*Eveniment: Click
THISFORM.RELEASE
```

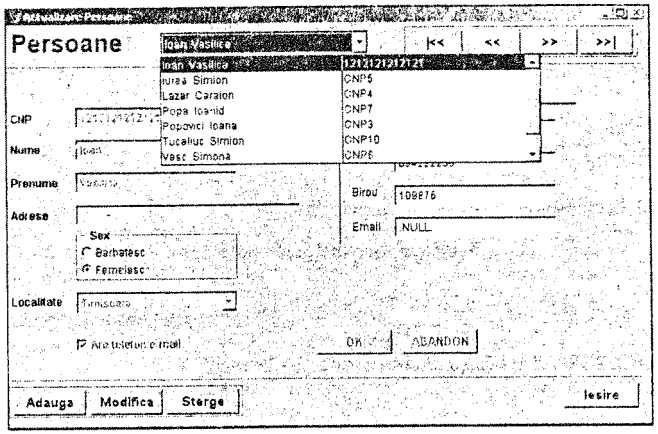


Figura 8.18. Formularul `frmPerscane` în faza de execuție

Lansarea în execuție a formularului (figura 8.18) se realizează cu ajutorul comenzii `DO FORM <numeformular>`:

```
DO FORM "\aplicatie\forms\frmpersoane"
```

Această comandă se include într-o secvență de cod ori se scrie direct în fereastra `Command`. Execuția se poate declanșa și cu butonul ! din bara de instrumente standard a Visual FoxPro, ori cu opțiunea `Form→Run Form` din meniul principal (sau rapid cu `Ctrl+E`). Aceste ultime opțiuni nu sunt accesibile decât dacă formularul este deschis în fereastra `Form Designer`. De asemenea, dacă dispunem de un proiect (tot pe parcursul realizării unei aplicații), se poate selecta numele formularului în secțiunea `Documents|Forms` a proiectului, folosind apoi pentru execuție butonul `Run`.

Formularele se pot lansa și *cu parametri*, dar numai prin comanda:

```
DO FORM numeformular [(parametrul,parametru2....)]
```

Reamintim că parametrii sunt preluați de evenimentul `Init` al formularului, deci în codul acestuia, pe prima linie, trebuie să existe declarațiile `PARAMETERS` sau `LPARAMETERS`. Parametrii sunt creați inițial ca variabile de tip `Logical`, cu valoarea implicită `.F.`, și vor rămâne așa dacă nu se transmite nimic prin comanda `DO FORM`.

Presupunând că, în funcție de anumite opțiuni, am vrea să lansăm formularul maximizat sau la dimensiunea „normală" (stabilită la proiectare), evenimentul `Init` trebuie modificat astfel:

Listing 8.3. Evenimentul `Init` în cazul formularelor parametrizate (varianta 1)

```
*Eveniment: Init
LPARAMETERS stare_formular
IF stare_formular="max" && "maximized"
    THISFORM.WindowState=2
ENDIF
THISFORM.cboPersoana.LISTINDEX=1      && obiectul va afișa implicit primul element din listă
THISFORM.cboPersoana.INTERACTIVECHANGE  && asigură "refresh-ul" controalelor legate
```

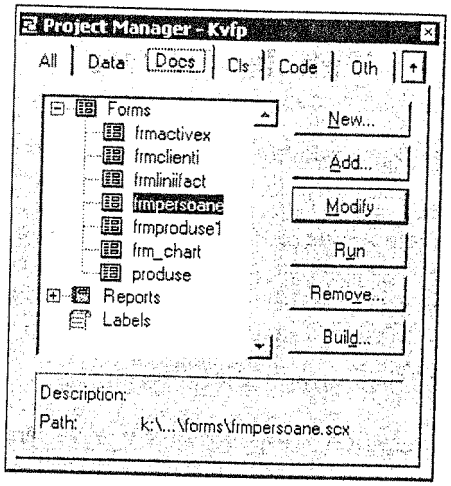



Figura 8.19. Formularele unui proiect în curs de realizare

Lansarea în execuție în noile condiții se poate face astfel:

```
DO FORM "\aplicatie\forms\frmpersoane.scx" WITH "max"
```

Totuși, dacă nu se folosește clauza opțională `WITH` sau dacă lansarea în execuție are loc prin opțiuni de meniu ori prin butonul `Run`, va apărea o eroare provenind din metoda `Init`. Într-adevăr, pentru proprietatea `WindowState` a unui formular, `WindowState=.F.` este o setare incorectă. Pentru a împiedica apariția erorii în cazul în care nu e nevoie de lansarea parametrizată, vom modifica din nou procedura evenimentului `Init` ca în listingul 8.4.

Listing 8.4. Evenimentul `Init` în cazul formularelor parametrizate (varianta 2)

```
LPARAMETERS stare_formular
* daca este de tip sir de caractere si are valoarea "max"
IF TYPE("stare_formular")="C" and LOWER(stare_formular)="max"
    THISFORM.WINDOWSTATE=2
ENDIF
THISFORM.CboPersoana.LISTINDEX=1
THISFORM.CboPersoana.INTERACTIVECHANGE
```

Se verifică mai întâi dacă s-a transmis ca parametru un șir de caractere și dacă acesta este „max", deoarece altă valoare nu interesează.

Notă. Un parametru se comportă ca o variabilă locală la nivelul procedurii evenimentului `Init`. Dacă el trebuie consultat altundeva în formular, în aceeași procedură `Init` se va atribui valoarea lui unei *variabile publice* sau, mai elegant, unei proprietăți a formularului, creată special.

Capitolul 9

Tabele virtuale

În mod obişnuit, tabelele virtuale sau derivate (există o anume diferenţă între aceşti termeni, diferenţă pe care o vom trece cu vederea) sunt percepute ca tabele noi, construite prin subconsultări aplicate asupra tabelelor de bază sau altor *view*-uri. Standardul SQL-92 defineşte *view*-urile ca tabele virtuale ce sunt materializate atunci când este invocat numele lor. Materializarea înseamnă, de fapt, execuţia frazei `SELECT`, ce constituie definiţia tabelei virtuale şi popularea cu înregistrări ce reprezintă rezultatul interogării.

9.1. Generalităţi despre tabele virtuale

Care ar fi motivele recurgerii la o tabelă derivată? În primul rând, pentru posibilitatea de a extrage, din una, două, trei... tabele, mai multe informaţii pe care să nu le stocăm nici în tabele propriu-zise (mari consumatoare de spaţiu), nici în variabile de memorie (atât de... volatile, dar şi limitate, la urma urmei). Deşi pare plauzibil, acest motiv prezintă serioase contraargumente. Dacă problema este numai de extragere şi afişare a datelor, atunci se recomandă utilizarea fie a cursoarelor, fie a fişierelor-interogare (filtre), ce au extensia `.QPR` şi pot fi lesne incluse în proiect. La urma urmei, `Query Designer` seamănă binişor cu `View Designer`. Problema este că fişierele-interogare nu fac, propriu-zis, parte din baza de date şi nu pot fi utilizate în `Data Environment` ale rapoartelor. Un alt atu al `views` faţă de `querys` în Visual FoxPro ţine de faptul că o tabelă derivată poate fi parametrizată (se poate solicita introducerea de la tastatură a valorii unui atribut al tabelei derivate chiar în momentul lansării, deci „filtrul" poate fi construit interactiv).

9.1.1. Ce sunt tabelele virtuale şi care este utilitatea lor

Noţiunea de tabelă derivată apare în BD relaţionale legată de nivelul extern, nivelul la care utilizatorii sau grupurile de utilizatori interacţionează cu baza de date. Din acest punct de vedere, este îndeobşte recunoscut faptul că tabela virtuală facilitează accesul la baza de date şi diminuează riscurile accesului neautorizat. În VFP putem spune că prin tabelele virtuale este facilitat accesul.

Cele mai importante avantaje ale tabelelor derivate decurg din faptul că acestea sunt actualizabile, iar modificarea lor poate fi propagată în tabelele-sursă din care provin. De asemenea, în Visual FoxPro, prin tabele derivate putem gestiona (cam mult spus) baze de date Oracle, SQL Server etc. Nu în ultimul rând, tabelele derivate sunt foarte utile în formulare, mai ales ca surse ale controalelor de tip `Grid`.

9.1.2. Tabele virtuale în SQL-92

Formatul general în SQL-92 al comenzii de creare a unei asemenea tabele virtuale este:

```
CREATE VIEW <nume-tabelă-virtuală> [<listă-coloane>] AS
    <frază SELECT>
    [WITH [<clauză de nivele>] CHECK OPTION]
```

unde

```
    <clauză de nivele> ::= CASCADED | LOCAL
```

O dată creată tabela virtuală, definiţia sa este salvată în schema bazei. Ulterior, ori de câte ori este necesar, la deschiderea/reîmprospătarea tabelei derivate, aceasta este (re)populată cu înregistrări extrase din cele ale tabelelor propriu-zise ce apar în clauza `FROM` a interogării. O tabelă virtuală poate fi deci privită şi ca o expresie de subconsultare a unei tabele persistente, stocată în bază şi invocată prin numele său.

Potrivit SQL-92, unei tabele virtuale nu i se pot asocia indecşi şi nici defini restricţii, deşi unele SGBD-uri optimizează lucrul cu *view*-urile folosind indecşii tabelelor persistente. Numele său este unic, şi nu se poate autoreferi, deşi un *view* poate fi creat pe baza unei combinaţii de tabele persistente şi/sau alte *view*-uri.

SQL-92 introduce şi noţiunea de *tabelă derivată*, definită asemănător celei virtuale, dar pentru care se pot defini restricţii şi asocia indecşi, fiind, spre deosebire de tabela virtuală, persistentă[30].

```
<subconsultare-tabelă> AS <nume-tabelă-virtuală> [<listă-coloane>]
```

La creare, o tabelă virtuală poate fi declarată actualizabilă sau non-actualizabilă (*Read-Only*). *Pentru a fi actualizabilă*, o tabelă virtuală *trebuie să aibă fiecare linie a sa asociată unei singure linii din tabelele bazei.* La modificarea unei linii din *view*, propagarea poate fi făcută, în aceste condiţii, fără probleme de ambiguitate.

Fireşte, o tabelă derivată de genul:

```
CREATE VIEW vJudete1 AS
  SELECT *
  FROM Judete
  WHERE regiune = 'Moldova' OR regiune = 'Dobrogea'
```

nu va crea probleme insolubile la actualizare. În schimb, deşi cu un conţinut identic VJUDETE1, tabela virtuală VJUDETE2, creată prin comanda care urmează, nu este actualizabilă nici în SQL-92, nici în majoritatea SGBD-urilor importante.

```
CREATE VIEW vJudete2 AS
  SELECT *
  FROM Judete
  WHERE regiune = 'Moldova'
UNION
  SELECT *
  FROM Judete
  WHERE regiune = 'Dobrogea'
```

30. Preluare din [Celko2000], p. 266.

Regulile actualizării tabelelor în SQL-92 sunt foarte stricte[31]:

1. Tabela virtuală trebuie derivată dintr-un SELECT cu o singură tabelă de bază; mai multe tabele de bază înseamnă mai multe niveluri ale tabelelor virtuale. Spre exemplu, pentru a construi o tabelă virtuală VCLIENTI în care pot fi modificate: denumirea clientului, adresa sa, denumirea localității în care își are sediul și numele județului, sunt necesare următoarele *view*-uri – listingul 9.1.

Listing 9.1. Tabele virtuale definite pe baza altor tabele virtuale

```
CREATE VIEW vJudete AS
  SELECT *
  FROM Judete

CREATE VIEW vLocalitati AS
  SELECT CodPost, Loc, vJudete.Jud, Judet, Regiune
  FROM LOCALITATI INNER JOIN vJUDETE
ON LOCALITATI.Jud = vJUDETE.Jud

CREATE VIEW vClienti AS
  SELECT CodCl, DenCl, Adresa, vLocalitati.CodPost, Loc, Jud, Judet, Regiune
  FROM CLIENTI INNER JOIN vLOCALITATI
ON CLIENTI.CodPost = vLOCALITATI.CodPost
```

2. Tabelele virtuale trebuie să includă toate coloanele cheilor primare/alternative ale tabelelor de bază. O linie dintr-o tabelă virtuală trebuie să corespundă unei singure linii din tabela de bază.
3. Toate coloanele neincluse în *view* trebuie să permită valori NULL sau să prezinte valori implicite. Altminteri, inserarea unei linii într-o tabelă derivată ar fi imposibilă.

Nu poate fi actualizată o tabelă virtuală creată prin fraze SELECT în care apar:

- clauze GROUP BY / HAVING,
- funcții agregat,
- coloane calculate,
- UNION,
- INTESECT,
- EXCEPT (MINUS),
- SELECT DISTINCT.

Din punctul de vedere al actualizării tabelelor virtuale, SQL-92 este mult mai limitat decât o cer regulile lui Codd, preferându-se siguranța. Multe SGBD-uri sunt însă mai îngăduitoare în această privință. Mai mult, în SQL3 și o serie de SGBD-uri, prin mecanisme de genul declanșatoarelor INSTEAD OF, multe probleme ale tabelelor virtuale, insolubile în SQL-92, își găsesc rezolvarea.

Atunci când tabela virtuală este actualizabilă, se poate folosi clauza WITH CHECK OPTION pentru un mai bun control al modificărilor. vJudeteMoldova conține înregistrările tabelei JUDETE pentru care valoarea atributului Regiune este Moldova:

31. [Celko99], p. 57


```
CREATE VIEW vJudeteMoldova AS
   SELECT *
   FROM judete
   WHERE Regiune='Moldova'
   WITH CHECK OPTION
```

Comanda INSERT în formatul:

```
INSERT INTO vJudeteMoldova VALUES ('BC', 'Bacau', 'Moldova')
```

se execută fără probleme, ceea ce nu este valabil și pentru:

```
INSERT INTO vJudeteMoldova VALUES ('AR', 'Arad', 'Banat'),
```

care va genera un mesaj de eroare din cauza regiunii care nu este acceptată în tabela virtuală, grație clauzei CHECK OPTION. În SQL-92, clauza de verificare este, implicit, declanșată în cascadă (CASCADED), adică atât pentru tabela virtuală curentă, cât și pentru cele pe baza cărora a fost construită, nivel cu nivel. Cu toate acestea, CASCADED se poate înlocui cu LOCAL, caz în care se verifică numai WHERE-ul prezentei tabele virtuale, fără a se testa modul în care modificările afectează tabelele derivate de pe nivelurile inferioare.

Ștergerea unei tabele viruale din schema bazei de date se realizează prin comanda DROP VIEW.

```
DROP VIEW <nume-tabelă-virtuală> <comportament>
```

unde

```
<comportament> ::= CASCADE | RESTRICT
```

Prin clauza CASCADE se șterg atât tabela virtuală curentă, cât și toate cele create pe baza acesteia, în timp ce RESTRICT interzice operațiunea atâta timp cât există măcar un *view* construit pe baza tabelei virtuale curente.

9.2. Crearea tabelelor derivate VFP în mod asistat

Visual FoxPro este un produs generos în ceea ce privește tabelele derivate care, după cum se va vedea în continuare, acordă o mare libertate dezvoltatorilor, în comparație cu restricțiile standardului SQL-92. În VFP tabelele virtuale pot fi create atât grafic, cât și prin program. Important este că pot fi definite explicit atributele modificabile sau nemodificabile, modul de actualizare a tabelelor, ba chiar mai mult, se pot defini indecși și reguli de validare chiar la nivel de *view*[32].

Făcând abstracție de atributele denormalizate introduse în capitolul 6 în tabela LINIIFACT, pentru început ne propunem să construim o tabelă virtuală ce conține liniile (detaliile) facturilor, prin parcurgerea procedurii de creare în mod asistat. Prin urmare, va trebui să adăugăm (de fapt, să creăm) în fereastra Project Manager un nou obiect de tip *view*. Acest lucru se realizează în felul următor:

- selectarea paginii Data (sau nodul Data din pagina All);
- expandarea nodului Databases, apoi a nodului corespunzător bazei de date (VINZARI);
- selectarea nodului Local views;
- „apăsarea" butonului New, deschizându-se fereastra New Local View;

32. Vezi și [Fotache&Strîmbei99-2].

- „acționarea", din fereastra `New Local View`, a butonului sub care se găsește eticheta `New view`.

Ca urmare a acestei succesiuni de operații, va fi lansat instrumentul `View Designer`. Procedura propriu-zisă de creare a tabelei virtuale poate fi descrisă astfel:

1. *Specificarea câmpurilor din tabelele-sursă din baza de date care vor forma schema finală a tabelei derivate*

Cum e și normal, înainte de a declara expresiile câmpurilor care vor forma schema tabelei derivate, `View Designer`-ul solicită specificarea tabelelor de bază din care se va construi tabela virtuală. Acest lucru se realizează prin fereastra `Add Table`, care, de obicei, apare implicit o dată cu lansarea asistentului pentru crearea unei noi tabele virtuale. Altminteri, această fereastră poate fi lansată folosind opțiunea `Add Table...` din meniul `Query`, meniu activ doar pe durata utilizării `View Designer`-ului sau `Query Designer`-ului (crearea unei interogări – `query` – se derulează la fel ca și crearea unei tabele virtuale, cu excepția pasului 4).

Lucrul cu fereastra `Add Table` este similar rapoartelor în ceea ce privește specificațiile legate de mediul de date (`Data Environment`). Singura excepție intervine atunci când între tabelele de bază nu există legături permanente pe baza cărora să fie deduse expresiile de joncțiune corespunzătoare; atunci utilizatorul este invitat prin dialogul `Join Condition` să precizeze ad-hoc aceste expresii.

Pentru exemplul nostru vor fi adăugate, în ordine, următoarele tabele: LINIIFACT, FACTURI, PRODUSE. Începând din momentul adăugării celei de-a doua tabele, în cadrul `Join` al `View Designer`-ului apar automat expresiile de joncțiune deduse pe baza legăturilor permanente deja declarate în baza de date (vezi figura 9.1).

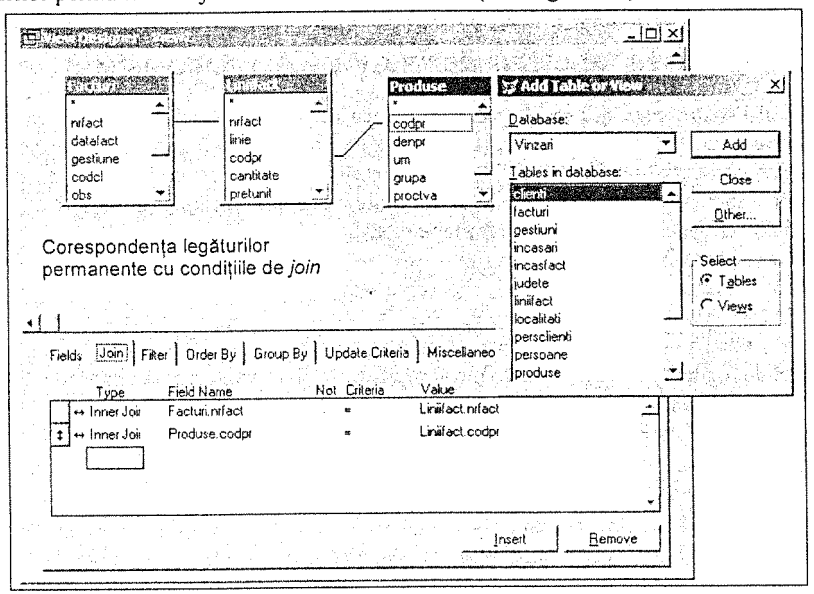


Figura 9.1. Adăugarea tabelelor de bază și formarea clauzelor de joncțiune

Pagina `View Designer`-ului în care se mută apoi „teatrul operațiunilor" este `Fields`. Astfel, din coloana `Available Fields` se vor selecta următoarele câmpuri: `LINIIFACT.nrfact`, `FACTURI.datafact`, `LINIIFACT.linie`, `LINIIFACT.codpr`, `PRODUSE.denpr`, `PRODUSE.um`, `LINIIFACT.cantitate`, `LINIIFACT.pretunit`, care se adaugă (pe rând sau în bloc) în coloana `Selected Fields` apelându-se la butonul `Add`.

Pentru adăugarea câmpurilor calculate dedicate *valorii fără TVA, TVA-ului și valorii totale a liniei din factură*, se va selecta căsuța de tip text `Functions and Expressions` în care, pentru valoarea fără TVA (spre exemplu), se va scrie direct expresia `LINIIFACT.cantitate * LINIIFACT.pretunit` (cei familiarizați cu instrumentul `Expresion Builder` îl pot apela prin butonul etichetat cu trei puncte de suspensie). În fine, pentru a aduce și acest câmp nou-creat în lista `Selected Fields`, vom apela din nou butonul `Add`.

Analog se procedează și pentru câmpurile prin care se calculează TVA-ul și valoarea totală a liniei din factură, folosind expresiile `LINIIFACT.cantitate* LINIIFACT.pretunit * PRODUSE.proctva` și `LINIIFACT.cantitate* LINIIFACT.pretunit*(1+PRODUSE.proctva)`[33].

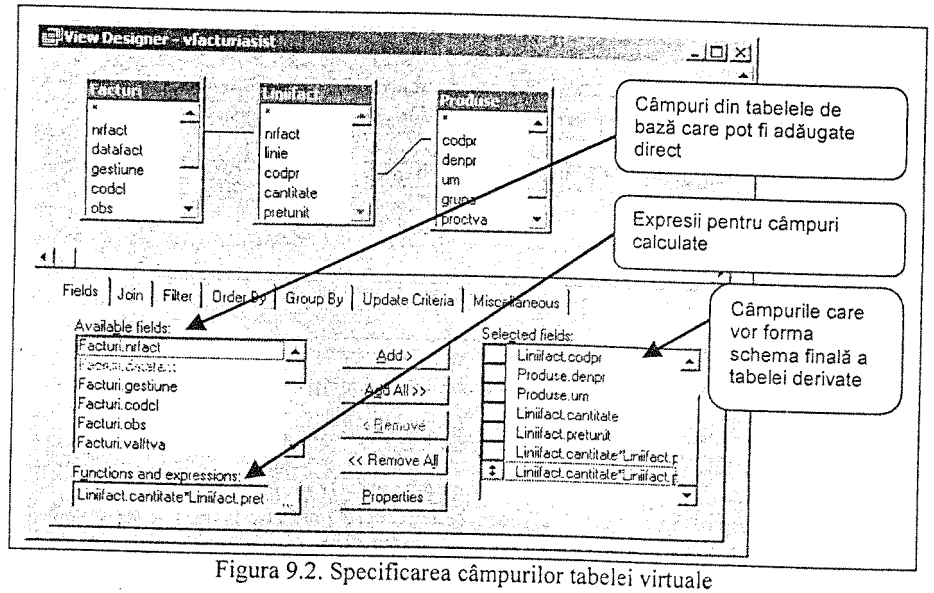


Figura 9.2. Specificarea câmpurilor tabelei virtuale

2. *Declararea eventualelor criterii de selecție pentru înregistrările care vor forma tabela derivată*

În această etapă se vor preciza criteriile prin care se filtrează setul de înregistrări constituit la deschiderea tabelei derivate. Spre exemplu, dacă dorim ca *view*-ul să conțină

33. În capitolul 6 paragraful 6.5.2 sunt exemplificate câteva proceduri stocate (mai exact triggere) pentru determinarea valorilor unor câmpuri calculate gen `FACTURI.ValTotala`. Pentru a expune câteva dintre caracteristicile utile ale *view*-urilor, am ignorat întru câtva această cale în prezentul capitol.

numai liniile facturilor emise între 01/08/2001 şi 07/08/2000, vom preciza un criteriu în felul următor: selectăm cadrul Filter, iar în prima linie a grid-ului acestuia alegem atributul FACTURI.Datafact din combo-box-ul aflat în coloana Field Name, apoi din coloana Criteria selectăm Between, iar în coloana Example vom introduce expresia CTOD("01/08/2000"), CTOD("07/08/2000"). Notaţia este un pic ciudată, dar e singura funcţionabilă, în ciuda recomandărilor din MSDN privitoare la criteriul Between pentru filtrele din View Designer. Pentru interpretarea corectă a datelor calendaristice precizate anterior este recomadabil să se execute o comandă gen SET DATE TO BRITISH înainte de deschiderea tabelei derivate.

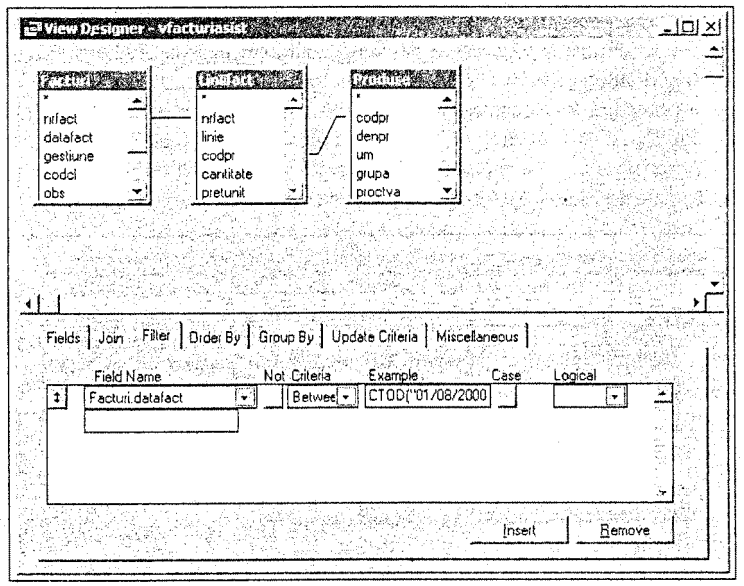


Figura 9.3. Declararea criteriilor de selecţie

3. *Declararea eventualelor criterii de ordonare şi de grupare a înregistrărilor care vor forma tabela derivată*

Pentru ordonarea înregistrărilor ce vor forma tabela derivată se foloseşte pagina Order By, unde în coloana Selected Fields se regăsesc câmpurile selectate în pasul 1. Se vor alege câmpurile LINIIFACT.nrfact şi LINIIFACT.linie din lista din stânga cadrului de pagină, adăugându-se apoi în lista Ordering Criteria prin selectarea butonului Add. În consecinţă, acestea vor deveni criterii de ordonare.

Dacă s-ar fi dorit constituirea înregistrărilor din tabela derivată din grupuri de înregistrări ale tabelelor de bază, s-ar fi utilizat cadrul de pagină Group By, în care câmpurile disponibile pentru grupare se regăsesc în lista Available Fields şi care, mutate în lista Grouped fileds, pot deveni criterii de grupare (dacă dorim să obţinem în final o tabelă derivată actualizabilă, atunci trebuie evitat acest cadru).

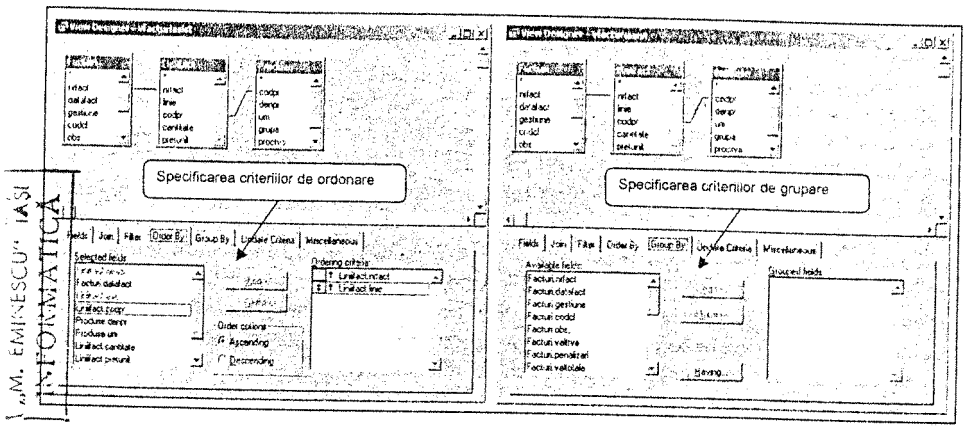


Figura 9.4. Precizarea criteriilor de ordonare şi grupare

4. *Specificarea opţiunilor privind gestiunea actualizărilor tabelelor de bază (normal, pentru situaţia în care tabela derivată este creată în acest scop)*

Dacă tabela derivată este construită numai pentru vizualizarea datelor aflate în tabelele de bază, acest pas nu este obligatoriu. Dacă se doreşte însă modificarea datelor din tabela derivată şi transmiterea acestor modificări în tabele de bază, atunci pasul 4 devine absolut obligatoriu.

În esenţă, această etapă presupune specificarea, folosind tab-ul Update Criteria, a următoarelor opţiuni:

Câmpurile ce vor fi declarate actualizabile

Spre deosebire de specificaţiile standardului SQL-92, în Visual FoxPro se poate forţa nota aşa încât să se încerce actualizarea mai multor tabele de bază folosind aceeaşi tabelă virtuală, cu rezultate, într-adevăr, destul de imprevizibile în anumite situaţii. În acest sens, în cadrul de pagină Update Criteria se găseşte o listă combinată numită Tables şi localizată undeva în stânga, prin care se filtrează o listă centrală. Aceasta va fi folosită pentru a face specificaţii fie asupra unei anumite tabele, fie asupra tuturor tabelelor de bază, privind câmpurile ce vor fi declarate actualizabile. În acest scop, câmpurile din tabelele de bază care se doresc a fi actualizabile prin tabela derivată se vor bifa în coloana deasupra căreia se află simbolul ce sugerează un creion.

Tot în lista centrală din pagina Update Criteria este foarte important să se precizeze câmpurile din tabelele de bază care formează cheia de actualizare (de regulă se mapează cu expresia cheii primare). Acest lucru se realizează bifând câmpurile corespunzătoare în coloana deasupra căreia se găseşte un simbol ce sugerează o cheie.

În cazul nostru, singura tabelă actualizabilă este LINIIFACT, aşa încât cheia de actualizare este alcătuită din atributele LINIIFACT.nrfact şi LINIIFACT.linie, care formează de fapt cheia primară a tabelei LINIIFACT. De asemenea, prin tabela derivată se declară actualizabile doar câmpurile LINIIFACT.nrfact, LINIIFACT.linie, LINIIFACT.codpr, LINIIFACT.cantitate şi LINIIFACT.pretunit (vezi figura 9.5).

Modul de propagare a modificărilor

Mecanismul pentru propagarea modificărilor din tabela derivată către tabele de bază se aseamănă cu execuția unei fraze UPDATE-SQL. Astfel, valorile care vor fi transmise în tabele de bază sunt preluate în clauza SET pornind de la câmpurile declarate actualizabile, însă clauza WHERE care efectuează testul prin care se determină liniile din tabelele de bază care vor prelua aceste valori poate fi constituită în mai multe moduri. În acest sens, în cadrul de pagină Update Criteria există un grup de butoane-radio etichetat SQL WHERE clauses includes (localizat în dreapta – vezi figura 9.5), cu următoarele opțiuni:

- Key fields only – prin care numai valorile câmpurilor declarate cheie sunt luate în considerare în clauza WHERE. Cu alte cuvinte, propagarea actualizărilor are loc numai în condițiile în care se constată că au rămas nemodificate valorile câmpurilor din tabelele de bază care au fost declarate cheie în tabela derivată;
- Key and updatable fields – prin care sunt luate în considerare atât câmpurile-cheie (vezi opțiunea precedentă), cât și cele declarate actualizabile. Dacă se constată că valorile din tabelele de bază ale câmpurilor-cheie sau actualizabile au fost modificate între timp de alt utilizator/instanță a aplicației, propagarea actualizărilor este oprită;
- Key and modified fields – prin care la actualizarea tabelelor de bază se verifică dacă, din momentul constituirii tabelei virtuale până în cel al propagării actualizărilor tabelei derivate, au fost operate modificări nu numai în atribute-cheie, dar și în orice alt atribut a cărui valoare a fost modificată în view, caz în care propagarea actualizărilor este blocată.

Prin aceste opțiuni se face de fapt verificarea (la momentul declanșării procesului de propagare a actualizărilor) stării înregistrărilor din tabelele de bază (stare consemnată în momentul „populării" tabelei derivate). Rolul câmpurilor-cheie este de a face posibilă maparea fiecărei înregistrări din tabela derivată la o singură înregistrare din tabela-sursă și pentru a diminua riscul incoerenței tabelelor în modul de lucru multiutilizator (vezi capitolul 13). Această verificare se realizează prin clauza WHERE a frazei SQL corespunzătoare. În exemplul nostru vom opta pentru a doua opțiune – Key and updatable fields.

Modul de efectuare a actualizărilor

Deși mecanismul clasic de propagare a modificărilor implică construirea și execuția unei fraze UPDATE, mai există și posibilitatea apelării la un alt mecanism de actualizare: mai întâi liniile ce trebuie actualizate se șterg, după care se reinserează cu noile valori provenite din tabela derivată. În acest sens, în cadrul de pagină Update criteria există un grup de butoane-radio etichetat Update using, cu două variante: SQL DELETE then INSERT și SQL UPDATE (vezi figura 9.5). Pentru cazul concret al exemplului nostru vom opta pentru a doua opțiune, și anume SQL UPDATE.

În fine, pentru ca modificările din tabela derivată să actualizeze într-adevăr tabela de bază, este necesară bifarea check-box-ului Send SQL Updates aflat în stânga-jos în cadrul Update Criteria (vezi figura 9.5).

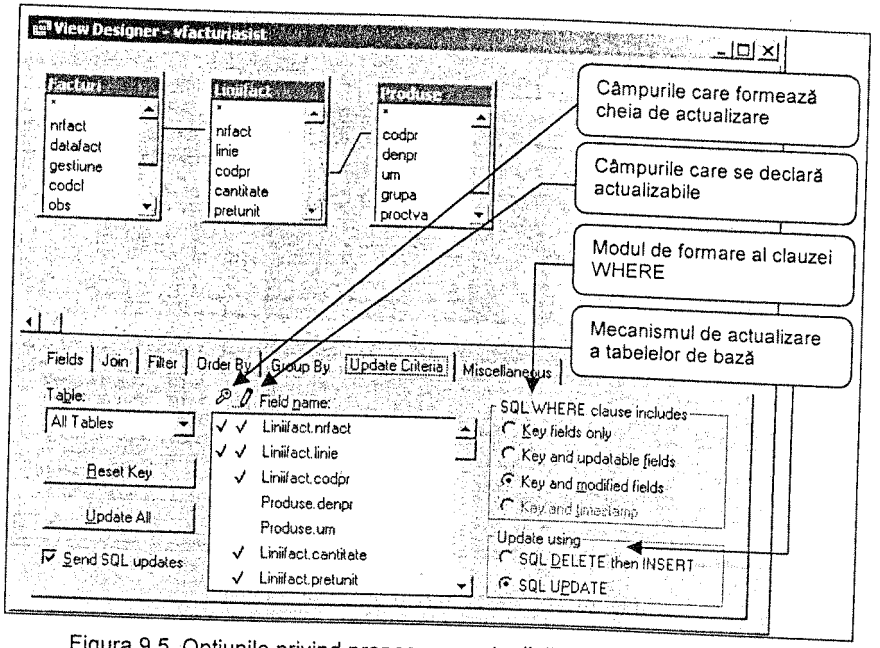


Figura 9.5. Opțiunile privind propagarea actualizărilor în tabelele de bază

În final, tabela virtuală se salvează sub un nume sugestiv și va fi inclusă în dicționarul bazei de date. O dată creată, această tabelă trebuie deschisă explicit prin comanda USE, operațiune în urma căreia are loc execuția frazei SELECT și popularea cu înregistrări corespunzătoare. De remarcat că împrospătarea tabelei derivate cu cele mai noi înregistrări ale tabelelor de bază nu necesită închiderea și deschiderea sa, acest lucru realizându-se prin instrucțiunea REQUERY(). Din păcate, această funcție nu poate fi inclusă în tranzacții.

9.3. Conflicte la propagarea modificărilor în tabelele de bază

Cei care lucrează frecvent cu tabele derivate în Visual FoxPro este aproape imposibil să nu fi avut parte de mesajul Update conflict (eroarea cu numărul 1585). Pentru a clarifica circumstanțele în care poate apărea această problemă, vom insista puțin asupra ei în acest paragraf, anticipând discuția din capitolul 13.

Situația cea mai plauzibilă când ar putea să apară eroarea 1585 este lucrul în rețea, atunci când doi utilizatori efectuează în același timp operații de actualizare pe aceleași înregistrări. De exemplu, pornind de la tabela derivată tocmai construită, presupunem că aceasta stă la baza unui formular prin care se permite modificarea detaliilor (liniilor/produselor) din facturi.

Să ne imaginăm următorul scenariu:

1) *Primul utilizator* accesează formularul pentru a modifica detaliile facturii 1122. Lansarea formularului de lucru al aplicației cu facturi presupune și deschiderea tabelei derivate, a cărei formare este ilustrată în figura 9.6.

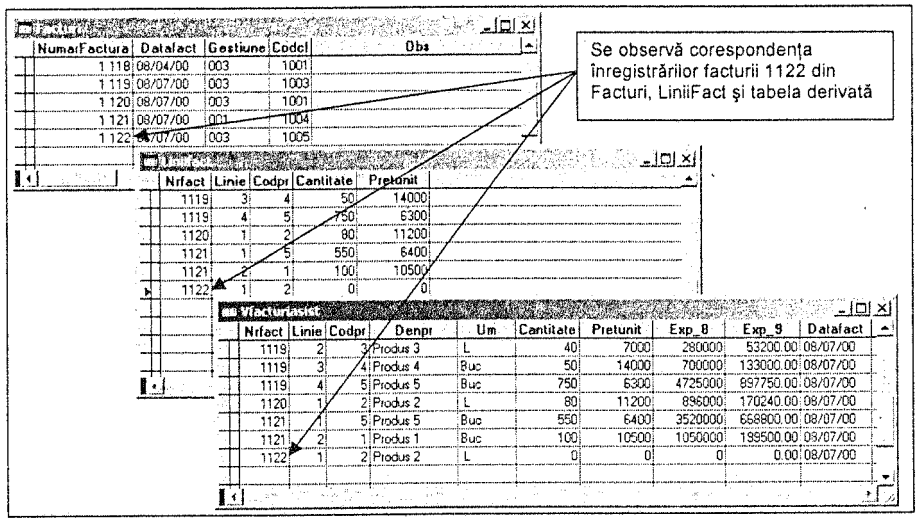


Figura 9.6. Starea inițială a tabelei derivate

2) *Al doilea utilizator* modifică aceeași factură – `1122` – la care primul utilizator încă lucrează, schimbând valoarea atributului `FACTURI.nrfact` în `1123`. Prin cascadă, datorită *trigger*-ului `trg_upd_facturi`, această modificare este propagată în tabela LINIIFACT asupra înregistrărilor corespunzătoare facturii vechi cu numărul `1122`, după cum se ilustrează în figura 9.7.

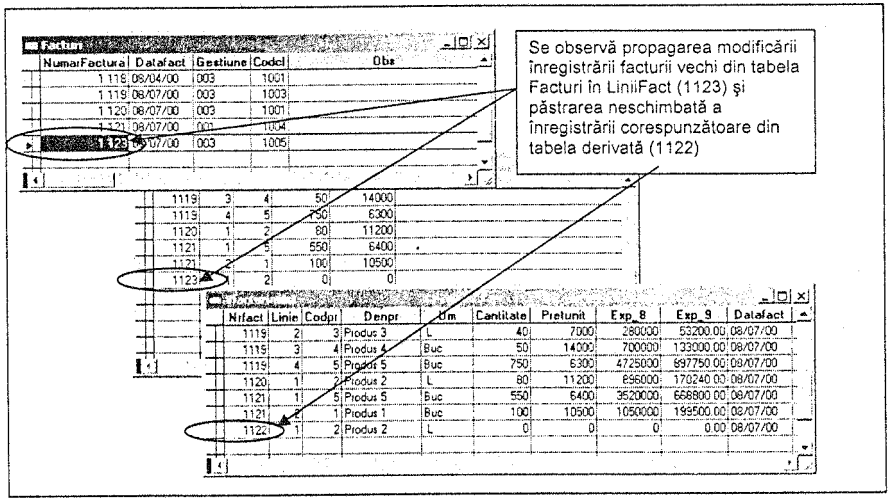


Figura 9.7. Propagarea modificării din tabela `FACTURI`

3) În fine, primul utilizator modifică prețul unitar și cantitatea și încearcă să valideze actualizările făcute. Ca urmare, va apărea fatalul mesaj `Update conflict` (vezi figura 9.8).

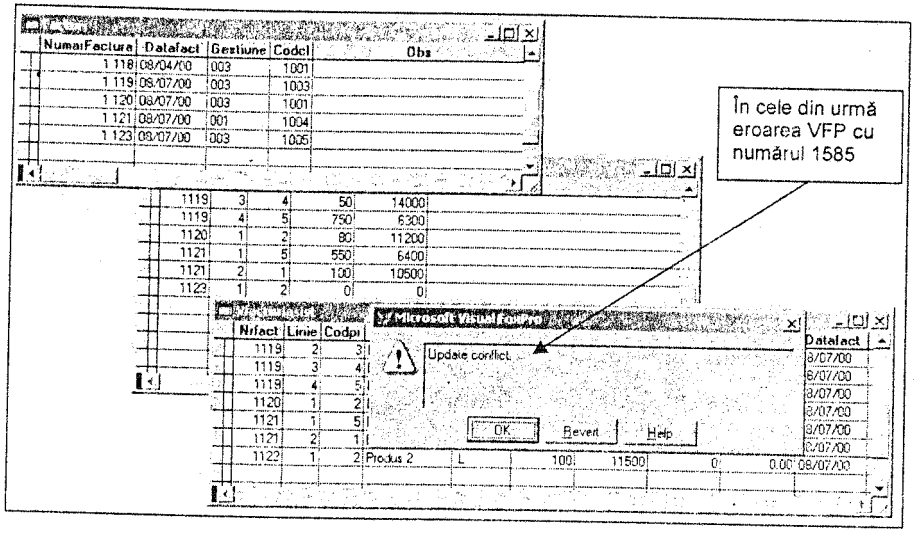


Figura 9.8. Generarea mesajului `Update conflict`

Explicația este următoarea: sistemul a încercat să propage modificările din tabela derivată în tabela de bază LINIIFACT, adică a încercat să actualizeze o înregistrare corespunzătoare facturii `1122`. Însă în tabela LINIIFACT această factură nu mai există (în locul său aflându-se factura `1123`), fapt semnalat de VFP prin mesajul `Update conflict`.

Eroarea se declanșează indiferent de specificațiile clauzei `SQL WHERE` declarate în `Update criteria`, deoarece `NrFact` este atribut-cheie.

Există însă și alte circumstanțe în care poate apărea această problemă în funcție de clauza `SQL WHERE` selectată. Astfel:

- în cazul `Key and Updatable Fields`, dacă al doilea utilizator modifică în tabela LINIIFACT oricare câmp declarat actualizabil în tabela derivată, adică `LINIIFACT.nrfact`, `LINIIFACT.linie`, `LINIIFACT.codpr`, `LINIIFACT.cantitate` sau `LINIIFACT.pretunit`, se va genera aceeași eroare `1585`. Să presupunem că al doilea utilizator modifică `LINIIFACT.cantitate` specificând valoarea `150`. Urmările sunt cele din figura 9.9.

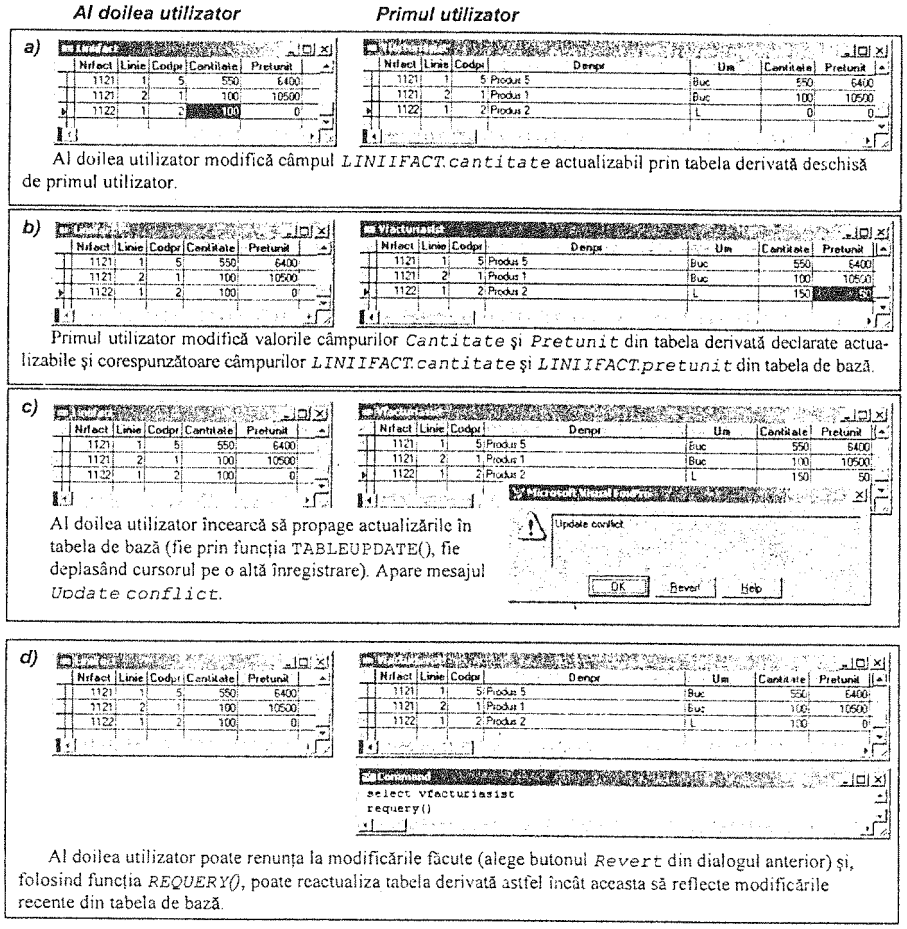


Figura 9.9. Un alt `Update conflict` și modul în care se reîmprospătează tabela derivată

- în cazul `Key and Modified Fields` – mesajul poate apărea și la modificarea unui câmp din tabela de bază, chiar dacă în tabela derivată câmpul corespunzător nu este cheie sau nu este declarat actualizabil.

Atenție! Trimiterea în tabela de bază a modificărilor operate în *view* se realizează, de regulă, prin simpla părăsire a înregistrării curente și plasarea pe altă înregistrare. O alternativă o reprezintă folosirea funcției `tableupdate()` executată din fereastra de comenzi. Dacă însă intervine eroarea `Update conflict`, tabela derivată pare blocată, modificările neputându-se „comite" în tabela de bază. Pentru a rezolva problema, fie se apasă butonul `Revert` din dialogul determinat de eroarea `Update conflict`, fie se execută din fereastra de comenzi funcția `tablerevert()`.

Alegerea clauzei `SQL WHERE` pentru actualizarea tabelei de bază este importantă din cel puțin două puncte de vedere: cel al performanțelor și cel al suprascrierii modificărilor efectuate de alți utilizatori asupra tabelei de bază. Din această perspectivă:

- Opțiunea `Key and Updatable Fields` presupune efectuarea unui test destul de laborios, în funcție de numărul de câmpuri declarate actualizabile în tabela derivată, însă nu ridică probleme de suprascriere a modificărilor efectuate de ceilalți utilizatori asupra tabelei de bază. Acest lucru se datorează faptului că modificările din câmpurile neactualizabile nu se propagă, iar dacă în tabela de bază câmpurile declarate actualizabile în tabela derivată au fost deja modificate, apare mesajul `Update conflict`, fiind împiedicată astfel suprascrierea acestora (după cum am exemplificat în figura 9.9).
- Opțiunea `KeyFields Only` presupune efectuarea unui test puțin costisitor, însă în situația în care un alt utilizator modifică câmpurile din tabela de bază declarate actualizabile în tabela derivată, există riscul ca aceste modificări să fie suprascrise, după cum se ilustrează în figura 9.10.

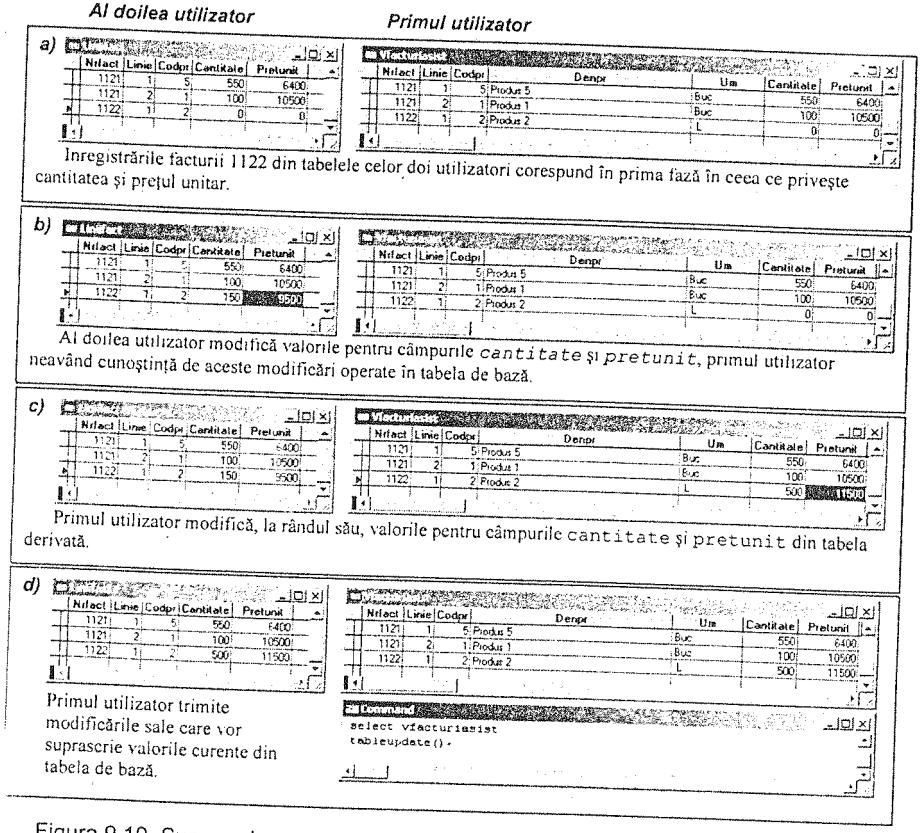


Figura 9.10. Suprascrierea valorilor din tabela de bază în cazul în care pentru tabela derivată a fost specificată opțiunea `Key Fields Only`

– Opțiunea `Key and Modified Fields` – presupune efectuarea unui test destul de greoi, în funcție de numărul de câmpuri care au fost modificate (indiferent dacă sunt actualizabile sau nu), și lasă deschisă posibilitatea suprascrierii modificărilor efectuate de alți utilizatori asupra câmpurilor din tabela de bază declarate actualizabile în tabela derivată. Acest lucru se întâmplă numai în cazul în care pentru efectuarea actualizărilor s-a recurs la varianta `SQL DELETE then INSERT` (vezi figura 9.11).

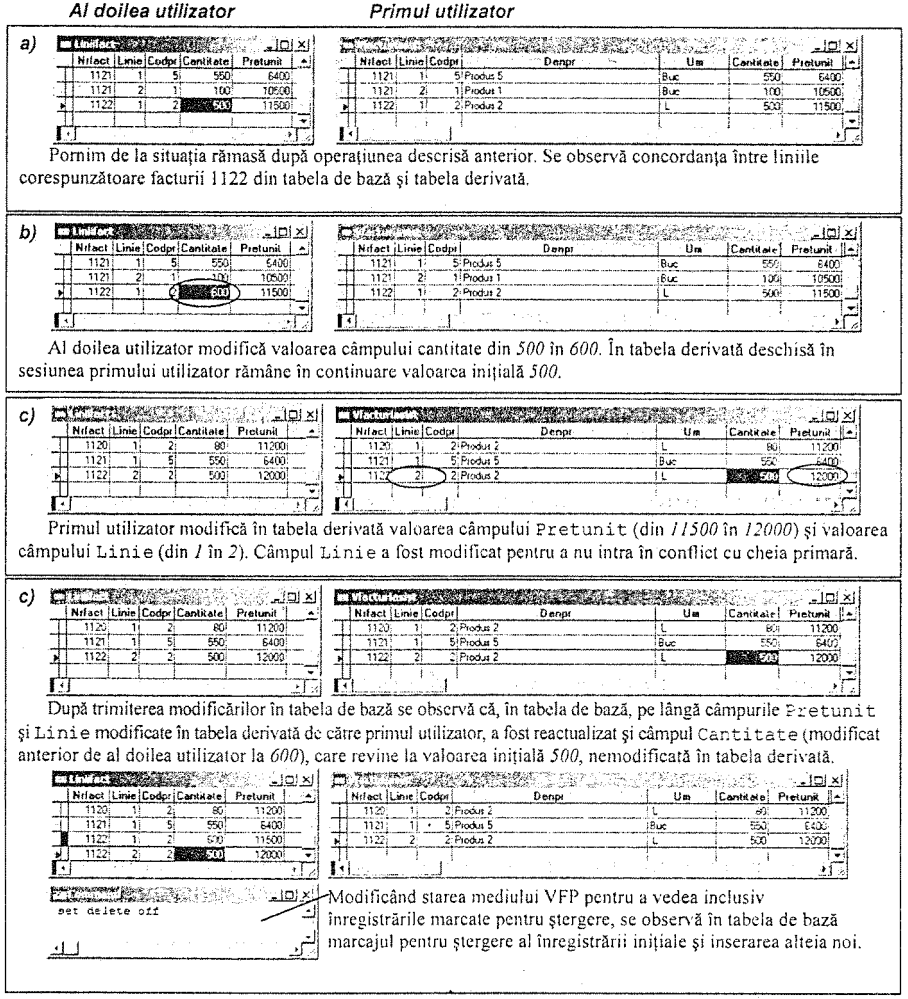


Figura 9.11. Suprascrierea valorilor din tabela de bază în cazul în care pentru tabela derivată a fost specificată opțiunea `Key and Modified Fields`

În acest ultim exemplu am presupus că primul utilizator modifică și valoarea câmpului `Linie`, pentru că la propagarea modificărilor înregistrarea veche din tabela de bază este

doar marcată pentru ștergere și nu ștearsă fizic. În consecință, putem încălca restricția de cheie primară.

9.4. Obținerea de informații privind structura tabelelor virtuale. Crearea *view*-urilor prin program

O tabelă derivată apare în baza de date ca obiect distinct ce poate fi localizat în `Project Manager` sub nodul `Local Views`. De fapt, după cum se poate observa din modul asistat exemplificat anterior, crearea unei tabele virtuale presupune două componente majore:

1 – mecanismul de definire a structurii și setului de înregistrări care formează tabela derivată. Acesta constă în execuția unei fraze `SQL-SELECT` rezultată din informațiile care se găsesc în cadrele de pagină `Fields`, `Join`, `Filter`, `Order by` și `Group by`, frază care, în modul `View Designer`, poate fi vizualizată prin opțiunea `Query` → `View SQL` din meniul principal VFP (vezi figura 9.12).

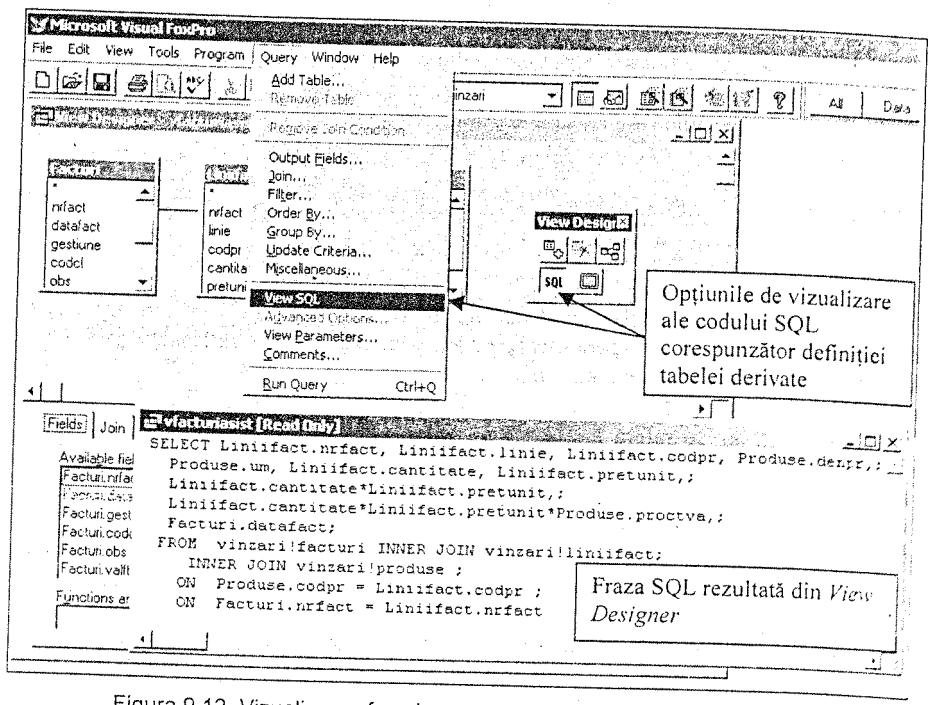


Figura 9.12. Vizualizarea frazei `SELECT-SQL` asociată tabelei derivate

2 – mecanismul de actualizare (dacă este cazul) a tabelelor pe baza modificărilor operate în tabela derivată, mecanism care are la bază specificațiile din cadrul `Update Criteria`.

9.4.1. Tabelele derivate în dicționarul de date

Atât fraza SELECT-SQL, cât și specificațiile care determină modul de actualizare a tabelei de bază se regăsesc în containerul bazei de date (.DBC) ca proprietăți ale unor obiecte de tip view și fields. Informațiile din dicționarul bazei de date pot fi accesate direct interogând fișierul cu extensia .DBC, dar, după cum am mai văzut în capitolul 6, modul de prezentare nu este supărător de inteligibil. Mai elegant, se pot folosi instrumentele mediului VFP, adică funcția DBGETPROP() pentru a „citi" informații din container și funcția DBSETPROP() pentru a modifica/actualiza dicționarul de date:

```
DBGETPROP(cName, cType, cProperty)
DBSETPROP(cName, cType, cProperty, ePropertyValue)
```

unde:

- cName – este numele obiectului (bază de date, tabelă, tabelă virtuală, atribut [câmp], conexiune) a cărui proprietate va fi consultată sau modificată;
- cType – reprezintă tipul obiectului (database, table, view, field, connection) a cărui proprietate va fi consultată sau modificată;
- cProperty – desemnează numele proprietății vizate (proprietățile, ca și numărul lor, sunt specifice fiecărui tip de obiect din containerul bazei de date);
- ePropertyValue – reprezintă, numai în cazul funcției DBSETPROP și numai pentru proprietățile care nu sunt declarate read-only, o expresie din care va rezulta noua valoare care va fi asignată proprietății obiectului vizat.

De exemplu, dacă dorim afișarea frazei SELECT-SQL care stă la baza unei tabele derivate, se poate recurge la comanda:

```
? DBGETPROP('vfacturiasist', 'view', 'sql')
```

Pentru o tabelă derivată, interesează valorile proprietăților asociate *view*-ului respectiv și ale câmpurilor (fields) componente. Din acest punct de vedere, principalele proprietăți accesibile prin funcțiile sus-amintite sunt:

- la nivelul *view*-ului:
  - SQL – se referă la fraza SQL care este executată la deschiderea tabelei derivate;
  - Tables – reprezintă o listă (a cărui delimitator este virgula) ce conține numele tabelelor-sursă;
  - UpdateType – reprezintă un număr (sau o directivă inclusă în Foxpro.h) care indică modul în care se vor efectua actualizările asupra tabelelor-sursă:
    - 1 sau DB_UPDATE – indică folosirea comenzii UPDATE, corespunzător opțiunii SQL Update din View Designer;
    - 2 sau DB_DELETEINSERT – desemnează mecanismul ștergere + inserarare corespunzător opțiunii SQL DELETE then INSERT din View Designer;
  - WhereType – se referă la un număr (sau o directivă inclusă în Foxpro.h) care indică cum se va forma testul corespunzător clauzei SQL WHERE. Astfel, dacă valoarea specificată pentru această proprietate este:

    - 1 sau DB_KEY – înseamnă că se va ține cont numai de câmpurile cheie, corespunzător opțiunii Key fields only din View Designer;
    - 2 sau DB_KEYANDUPDATABLE – înseamnă că se va ține cont de câmpurile cheie plus câmpurile declarate actualizabile, corespunzător opțiunii Key and updatable fields din View Designer;
    - 3 sau DB_KEYANDMODIFIED – înseamnă că se va ține cont de câmpurile declarate chei plus toate câmpurile ale căror valori inițiale au fost modificate, corespunzător opțiunii Key and modified fields din View Designer;
    - 4 sau DB_KEYANDTIMESTAMP – înseamnă că se va ține cont de câmpurile declarate cheie și de marca temporală (în cazul în care tabelele acceptă o marcă temporală).
  - SendUpdates – acționează ca un comutator care activează sau dezactivează actualizarea tabelelelor pe baza modificărilor survenite în *view* – corespunde opțiunii Send SQL Updates din View Designer.

- la nivelul câmpurilor (fields) tabelei derivate:
  - KeyField – conține o valoare de logic (TRUE/FALSE) prin care se indică dacă respectivul câmp este declarat cheie (.T.), similar bifării check-box-ului corespunzător din prima coloană a listei de câmpuri din cadrul Update Criteria al View Designer-ului;
  - UpdateName – desemnează numele câmpului din tabela-sursă la care se mapează câmpul din tabela derivată (poate reprezenta destinația propagării actualizării). Această proprietate este importantă în cazul în care numele câmpului diferă în tabela derivată față de tabela-sursă;
  - Updatable – conține .T. dacă respectivul câmp este declarat actualizabil, similar bifării check-box-ului corespunzător din a doua coloană a listei de câmpuri din cadrului Update Criteria al View Designer-ului.

9.4.2. Crearea „programatică" a tabelelor virtuale

Aplicațiile profesionale dezvoltate în VFP ce folosesc tabele virtuale utilizează programe în care sunt prezente comanda CREATE SQL VIEW și o serie de variante ale funcției DBSETPROP(). Deși complicată la prima vedere, funcția DBSETPROP() este foarte puternică și prezintă atuul flexibilității, după cum se observă în listingul 9.2, în care este prezentat programul de creare a unei tabele derivate similare celei create în mod asistat.

Listing 9.2. Program Visual FoxPro 6 de creare a unei tabele derivate

```
*** CREAREA TABELEI DERIVATE vFacturi
#INCLUDE foxpro.h

IF !DBUSED('vinzari')
  OPEN DATABASE vinzari
ENDIF
SET DATABASE TO vinzari

IF USED('vFacturi')
  SELECT vFacturi
  USE
ENDIF

* Fraza SQL de creare a tabelei virtuale
CREATE SQL VIEW vFacturi AS ;
  SELECT LF.NrFact, F.DataFact, Linie, LF.CodPr, DenPr, ;
      UM, Cantitate, PretUnit, Cantitate * PretUnit AS ValFaraTVA, ;
          Cantitate * PretUnit * ProcTVA AS TVA, ;
          Cantitate * PretUnit * (1 + ProcTVA) AS ValTotala ;
  FROM LINIIFACT LF INNER JOIN FACTURI F ON LF.NrFact=F.NrFact ;
      INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
  ORDER BY LF.NrFact, Linie

* Pe baza tabelei derivate, se va actualiza numai LINIIFACT
DBSETPROP('vFacturi', 'View', 'Tables', 'LINIIFACT')

* Se declara atributele cheie (primara)
DBSETPROP('vFacturi.NrFact', 'Field', 'KeyField', .t.)
DBSETPROP('vFacturi.Linie', 'Field', 'KeyField', .t.)

* Eventual se declarara modul de mapare a atributelor
DBSETPROP('vFacturi.NrFact', 'Field', 'UpdateName', 'LINIIFACT.nrfact')
DBSETPROP('vFacturi.Linie', 'Field', 'UpdateName', 'LINIIFACT.linie')
**** etc

* Se declara ca actualizabile atributele cheie primara
DBSETPROP('vFacturi.NrFact', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Linie', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Codpr', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Cantitate', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Pretunit', 'Field', 'Updatable', .t.)

* Tipul modificarilor (UPDATE, nu INSERT-DELETE)
DBSETPROP('vFacturi', 'View', 'UpdateType', DB_UPDATE)

* La propagarea actualizarii in tabelele de baza, se verifica daca au
* intervenit modificari in continutul atributelor cheie si a celor actualizabile
DBSETPROP('vFacturi', 'View', 'WhereType', DB_KEYANDUPDATABLE)

* Semnalul final pentru propagarea modificarilor din tabela derivata in cea de baza
DBSETPROP('vFacturi', 'View', 'SendUpdates', .t.)
```

Pentru afişarea tabelelor derivate din baza de date se poate apela la interogarea:

```
SELECT * FROM vinzari.dbc WHERE OBJECTTYPE='View'
```

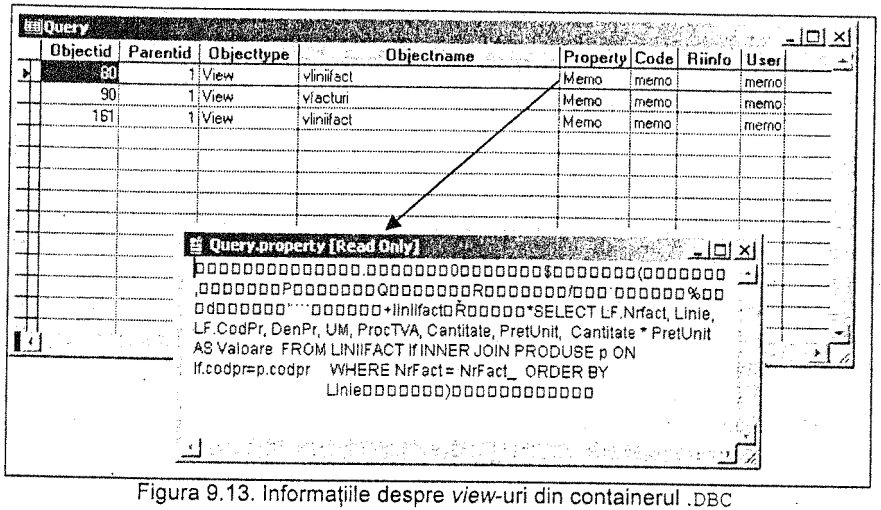



Figura 9.13. Informaţiile despre *view*-uri din containerul .DBC

Atributul `Property` conţine fraza `SELECT` pe baza căreia a fost construită tabela virtuală. Cât despre obţinerea datelor despre atributele tabelelor derivate, iată interogarea care generează rezultatul din figura 9.14:

```
SELECT v2.objectname, v1.objectid, v1.objectname, ;
  left(v1.property, 200) ;
FROM vinzari.dbc v1 INNER JOIN vinzari.dbc v2 ;
  ON v1.parentid=v2.objectid ;
WHERE v2.OBJECTTYPE='View' AND v1.OBJECTTYPE='Field' ;
ORDER BY v1.objectid
```

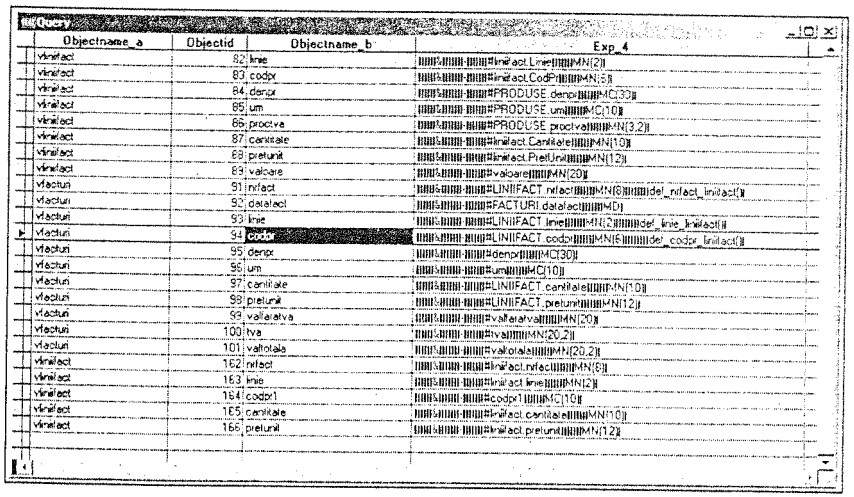


Figura 9.14. Informaţii despre câmpurile din definiţia tabelelor derivate din containerul .DBC

9.5. Alte considerații privind tabelele derivate în Visual FoxPro

Dacă până acum ne-am ocupat aproape exclusiv de înțelegerea mecanismelor specifice VFP care stau la baza creării tabelelor virtuale, în continuare vom încerca să punem în valoare câteva aspecte ale utilității practice a acestora. Plecând de la tabela derivată proaspăt-creată, vor fi operate câteva facilități suplimentare, pentru care se va face apel la alte proprietăți ale tabelelor derivate în VFP.

Una dintre cele mai tentante utilizări ale tabelei derivate VFACTURI ține de legarea sa la un control de tip rețea (*grid*) într-un formular, control care să permită utilizatorilor să completeze sau să modifice detaliile facturilor. Pentru început, să recapitulăm chestiunile esențiale ale VFACTURI:

- tabelele enumerate în fraza `SELECT` care contribuie la formarea setului de înregistrări al tabelei derivate sunt: FACTURI, LINIIFACT, PRODUSE;
- dintre aceste tabele, numai una recepționează eventualele modificări ale tabelei virtuale – LINIIFACT –, atributele `nrfact`, `linie`, `codpr`, `cantitate` și `pretunit`; celelalte două sunt folosite doar la popularea *view*-ului.
- câmpurile-cheie pentru actualizare sunt `nrfact` și `linie`;
- mecanismul de actualizare este de tip `DB_UPDATE`, iar la propagarea actualizărilor în tabela de bază sunt avute în vedere câmpurile-cheie și cele declarate actualizabile (`DB_KEYANDUPDATABLE`).

Pentru creșterea funcționalității tabelei derivate, pot fi identificate câteva caracteristici suplimentare și anume:

a) comportamentul tabelei derivate la adăugarea unei noi linii să fie asemănător cu cel al tabelei-sursă LINIIFACT, adică:
   - valoarea implicită pentru câmpul `nrfact` să fie numărul ultimei facturi din tabela FACTURI;
   - valoarea implicită pentru câmpul `linie` să fie consecutivă față de ultima valoare pentru acest câmp în cadrul aceleiași facturi.

b) La adăugarea unei noi linii, atributul `datafact` (neactualizabil) să afișeze data facturii corespunzătoare, valoare care se găsește în tabela FACTURI.

c) Modificarea valorii pentru câmpul `codpr` să antreneze și modificarea valorilor câmpurilor `denpr`, `UM` (neactualizabile) cu denumirea produsului și unitatea de măsură corespunzătoare din tabela PRODUSE. În plus, aceeași modificare a `codpr` să producă și resetarea valorilor (aducerea la 0) pentru câmpurile `cantitate` și `pretunit`, precum și pentru câmpurile calculate TVA, `ValfaraTVA`, `ValTotala`.

d) Modificarea valorii câmpurilor `cantitate` și `pretunit` să antreneze recalcularea automată a valorilor pentru câmpurile TVA, `ValfaraTVA`, `ValTotala`.

e) Blocarea validării unei linii, dacă nu s-a precizat nici un produs sau nici o valoare pentru cantitate și preț unitar.

Vom aborda gradat aceste probleme. Valorile implicite pentru câmpurile `nrfact` și `linie` din tabela virtuală pot fi determinate de aceleași proceduri stocate folosite pentru

câmpurile corespunzătoare din tabela LINIIFACT, și anume `def_nrfact_liniifact()` și `def_linie_linifact()`. Asocierea acestor proceduri la cele două atribute ale tabelei virtuale este o operațiune simplă, realizată prin intermediul proprietăților de tip `DefaultValue`:

```
DBSETPROP('vFacturi.NrFact', 'Field', 'DefaultValue',
'def_nrfact_liniifact()')
DBSETPROP('vFacturi.Linie', 'Field', 'DefaultValue',
'def_linie_liniifact()')
```

Dacă problema valorilor implicite pentru câmpurile `nrfact` și `linie` este relativ simplă, pentru câmpul `datafact` lucrurile se complică puțin. În acest scop, va trebui să construim o procedură stocată suplimentară care să returneze data facturii căreia îi corespunde noua linie din tabela derivată. O soluție pentru o astfel de procedură este prezentată în listingul 9.3.

Listing 9.3. Exemplu de regulă de validare la nivel de atribut al unei tabele virtuale

```
*-----------------------
PROCEDURE def_datafact_vfacturi
IF indexseek(vfacturi.nrfact, .T., 'facturi', 'nrfact')
  return facturi.datafact
ENDIF
ENDPROC
```

Se are în vedere faptul că procedura `def_nrfact_liniifact()`, care furnizează valoarea implicită pentru atributul `nrfact`, se execută anterior procedurii `def_datafact_vfacturi()`. Numele acestei proceduri se asociază proprietății `DefaultValue` pentru câmpul `DataFact`:

```
DBSETPROP('vFacturi.Datafact', 'Field', 'DefaultValue',
'def_datafact_vfacturi()')
```

Pentru recalcularea câmpurilor `TVA`, `ValfaraTVA` și `ValTotala` ca urmare a modificării cantității sau prețului unitar, vom adăuga un câmp nou în tabela derivată `ProcTva` care provine din tabela PRODUSE. Acest câmp este necesar pentru simplificarea calculelor. Atributele `TVA`, `ValfaraTVA` și `ValTotala` au un rol pur informativ (nu pot fi declarate actualizabile), însă pentru ca linia respectivă să fie corectă, se impune recalcularea acestora la fiecare modificare a unui atribut „sursă" (`cantitate`, `pretunit`). În acest scop, vom construi procedura stocată `calcul_val_lnfact()` din listingul 9.4.

Listing 9.4. Procedură stocată ce (re)calculează valorile fără TVA, cu TVA și totală în tabela virtuală `vFacturi`

```
*-----------------------
PROCEDURE calcul_val_lnfact
IF !EMPTY(vfacturi.cantitate) and !EMPTY(vfacturi.pretunit)
  replace vfacturi.ValFaraTVA with vfacturi.Cantitate * vfacturi.PretUnit
  replace vfacturi.TVA with vfacturi.cantitate * vfacturi.Pretunit * vfacturi.proctva
  replace vfacturi.valtotala with vfacturi.Cantitate * vfacturi.PretUnit * (1 + vfacturi.ProcTVA)
ENDIF
ENDPROC
*-----------------------
```

Această procedură va fi declarată regulă de validare la nivel de câmp pentru atributele `Cantitate` şi `PretUnit`, prin intermediul proprietăţii `RuleExpression`:

```
DBSETPROP('vFacturi.Cantitate', 'Field', 'RuleExpression',
   'calcul_val_lnfact()')
DBSETPROP('vFacturi.PretUnit', 'Field', 'RuleExpression',
   'calcul_val_lnfact()')
```

În scopul asigurării în permanenţă a corespondenţei între valorile câmpurilor `DenPr`, `UM` şi `ProcTVA` cu produsul specificat prin valoarea câmpului `CodPr`, este necesară o nouă procedură stocată având rolul de a actualiza respectivele câmpuri:

Listing 9.5. Procedură-regulă de validare pentru reîmprospătarea atributelor `DenPr`, `UM` şi `ProcTVA` din tabela derivată

```
*-------------------------------
procedure seek_codpr
*-------------------------------
*** Procedura de validare pentru cimpul CODPR din view-ul vfacturi
Parameter codpr_
*** Caut datele produsului indicat
IF indexseek(codpr_, .T., 'produse', 'codpr')
  replace vfacturi.denpr with produse.denpr
  replace vfacturi.um with produse.um
  replace vfacturi.procTVA with produse.proctva
*** Resetarea celorlalte câmpuri
  replace vfacturi.cantitate with 0
  replace vfacturi.pretunit with 0
  replace vfacturi.ValFaraTVA with 0
  replace vfacturi.TVA with 0
  replace vfacturi.valtotala with 0
ELSE
   messagebox('Atentie! Nu exista nici un produs avand codul '+ allt(str(codpr_))+'!',;
   16, 'Eroare cod produs ...')
  RETURN .F.
ENDIF
endproc
*-------------------------------
```

Această procedură este declarată regulă de validare care, după cum se observă din listingul anterior, mai îndeplineşte două sarcini: avertizarea utilizatorului în cazul unui cod de produs invalid şi resetarea valorii pentru câmpurile `cantitate`, `pretunit` şi pentru câmpurile calculate. Declararea acestei proceduri ca regulă de validare pentru câmpul `Codpr` se face prin instrucţiunea-funcţie:

```
DBSETPROP('vFacturi.Codpr', 'Field', 'RuleExpression',
   'seek_codpr(vfacturi.codpr)')
```

În ceea ce priveşte valorile câmpurilor `codpr`, `cantitate` şi `pretunit`, validarea unei linii din tabela derivată se poate face printr-o expresie evaluată la „părăsirea" înregistrării curente, declarată prin intermediul proprietăţii `RuleExpression`. Eventualul mesaj care ar trebui să apară la încălcarea restricţiei se precizează prin intermediul proprietăţii `RuleText`.

```
DBSETPROP('vFacturi',   'View','RuleExpression',   'IIF(empty
(Codpr) OR empty(cantitate) OR empty(PretUnit), .f., .t.)')
```



```
DBSETPROP('vFacturi',   'View','RuleText',   'Linia   nu   este
completata cu toate datele necesare !')
```

În practică ar mai fi util de rezolvat şi o problemă de genul: la deschiderea tabelei derivate, utilizatorul să precizeze interactiv facturile cărei luni să fie afişate. Această situaţie se referă, de fapt, la *parametrizarea tabelelor derivate*, operaţiune posibilă atât asistat, cât şi prin program.

În mod asistat, se va apela opţiunea `Query → View Parameters...` din meniul principal VFP, după ce în prealabil tabela derivată a fost deschisă în `View Designer`. Ca urmare a selectării acestei opţiuni, se va afişa o fereastră de editare a parametrilor. În gridul corespunzător se adugă un nou parametru care va fi numit `luna_`, de tip `numeric`. După părăsirea ferestrei `View Parameters` se selectează cadrul `Criteria`, unde se va adăuga un criteriu în care pentru `Field Name` se va specifica expresia `MONTH(FACTURI.datafact)`, în timp ce `Criteria` va fi „=", iar pentru `Example` se va preciza numele parametrului definit anterior plus semnul „?".

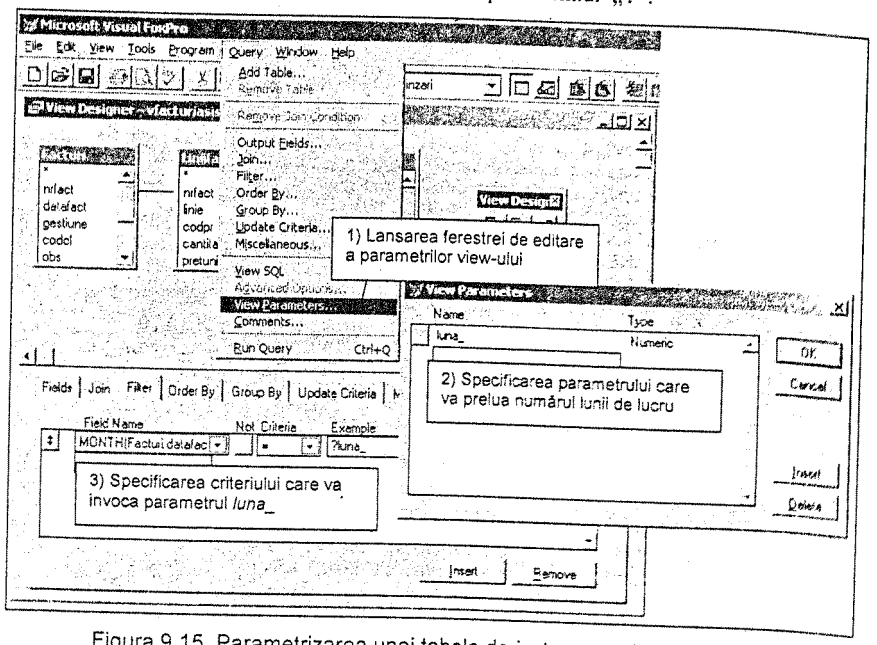


Figura 9.15. Parametrizarea unei tabele derivate create în mod asistat

Pentru *view*-ul VFACTURI pe care l-am creat folosind un script, parametrizarea acestuia determină o mică modificare a frazei `SELECT-SQL`, astfel:

```
CREATE SQL VIEW vFacturi AS ;
  SELECT LF.NrFact, F.DataFact, Linie, LF.CodPr, DenPr, ;
    UM, Cantitate, PretUnit, ;
    Cantitate * PretUnit AS ValFaraTVA, ;
    ProcTVA, Cantitate * PretUnit * ProcTVA AS TVA, ;
    Cantitate * PretUnit * (1 + ProcTVA) AS ValTotala ;
```

```
FROM LINIIFACT LF INNER JOIN FACTURI F ;
     ON LF.NrFact=F.NrFact ;
   INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
WHERE MONTH(F.datafact) = ?'luna_';
ORDER BY LF.NrFact, Linie
```

Prin urmare, noul script de definire a tabelei derivate va arăta ca în listingul 9.6.

Listing 9.6. O nouă versiune a programului de creare a `vFacturi`

```
*** CREAREA TABELEI DERIVATE vFacturi

#INCLUDE foxpro.h

IF !DBUSED('vinzari')
  OPEN DATABASE vinzari
ENDIF
SET DATABASE TO vinzari

IF USED('vFacturi')
  SELECT vFacturi
  USE
ENDIF

* Fraza SQL de creare a tabelei virtuale
CREATE SQL VIEW vFacturi AS ;
  SELECT LF.NrFact, F.DataFact, Linie, LF.CodPr, DenPr, ;
      UM, Cantitate, PretUnit, Cantitate * PretUnit AS ValFaraTVA, ;
          ProcTVA, Cantitate * PretUnit * ProcTVA AS TVA, ;
          Cantitate * PretUnit * (1 + ProcTVA) AS ValTotala ;
  FROM LINIIFACT LF INNER JOIN FACTURI F ON LF.NrFact=F.NrFact ;
      INNER JOIN PRODUSE P ON LF.CodPr=P.CodPr ;
  WHERE MONTH(F.datafact) = ?'luna_';
  ORDER BY LF.NrFact, Linie

* Pe baza tabelei derivate, se va actualiza numai LINIIFACT
DBSETPROP('vFacturi', 'View', 'Tables', 'LINIIFACT')

* Se declara atributele cheie (primara)
DBSETPROP('vFacturi.NrFact', 'Field', 'KeyField', .t.)
DBSETPROP('vFacturi.Linie', 'Field', 'KeyField', .t.)

* Eventual se declarara modul de mapare a atributelor
DBSETPROP('vFacturi.NrFact', 'Field', 'UpdateName', 'LINIIFACT.nrfact')
DBSETPROP('vFacturi.Linie', 'Field', 'UpdateName', 'LINIIFACT.linie')
**** etc

* Se declara ca actualizabile atributele cheie primara
DBSETPROP('vFacturi.NrFact', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Linie', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Codpr', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Cantitate', 'Field', 'Updatable', .t.)
DBSETPROP('vFacturi.Pretunit', 'Field', 'Updatable', .t.)

*  Se declara valori implicite pentru cateva campuri actualizabile
DBSETPROP('vFacturi.NrFact', 'Field', 'DefaultValue', 'def_nrfact_liniifact()')
DBSETPROP('vFacturi.Linie', 'Field', 'DefaultValue', 'def_linie_liniifact()')
DBSETPROP('vFacturi.Datafact', 'Field', 'DefaultValue', 'def_datafact_vfacturi()')
```



```
* Se declara cateva reguli
DBSETPROP('vFacturi.Codpr', 'Field', 'RuleExpression', 'seek_codpr(vfacturi.codpr)')
DBSETPROP('vFacturi.Cantitate', 'Field', 'RuleExpression', 'calcul_val_Infact()')
DBSETPROP('vFacturi.PretUnit', 'Field', 'RuleExpression', 'calcul_val_Infact()')

* Se declara regula de validare la nivel de inregistrare
DBSETPROP('vFacturi', 'View','RuleExpression', 'IIF(empty(Codpr) OR empty(cantitate) OR
empty(PretUnit), .f., .t.)')
DBSETPROP('vFacturi', 'View','RuleText', 'Linia nu este completata cu toate datele necesare !')


* Tipul modificarilor (UPDATE, nu INSERT-DELETE)
DBSETPROP('vFacturi', 'View', 'UpdateType', DB_UPDATE)

* La propagarea actualizarii in tabelele de baza, se verifica daca au
* intervenit modificari in continutul atributelor cheie si a celor actualizabile
DBSETPROP('vFacturi', 'View', 'WhereType', DB_KEYANDUPDATABLE)

* Semnalul final pentru propagarea modificarilor din tabela derivata in cea de baza
DBSETPROP('vFacturi', 'View', 'SendUpdates', .t.)
```

La deschiderea unei tabele derivate parametrizate, se va deschide mai întâi o fereastră de dialog (sau mai multe) care va solicita din partea utilizatorului o valoare validă pentru parametrul (parametrii) definit (definiți):

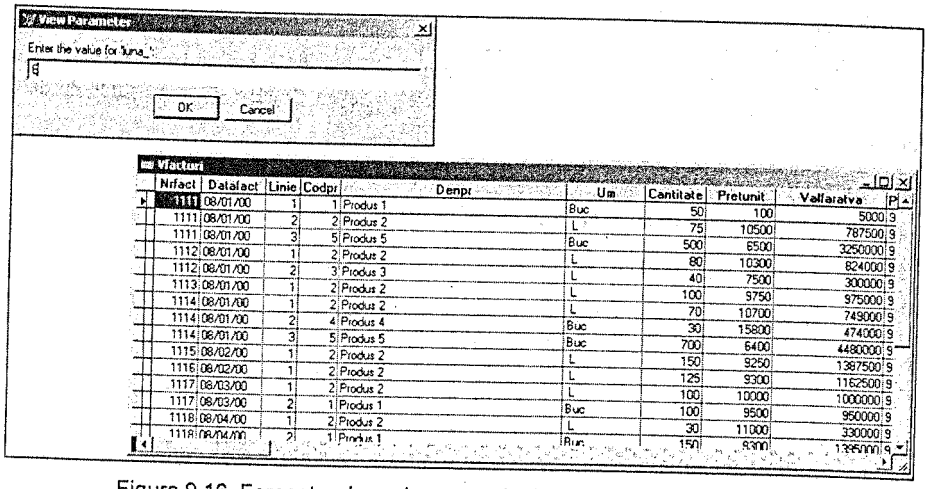


Figura 9.16. Fereastra de preluare a valorii pentru parametrul și setul de înregistrări rezultat după deschiderea *view*-ului

Capitolul 10

Formulare complexe

Vom încerca în acest capitol să prezentăm câteva căi de realizare a unei interfețe-utilizator ceva mai sofisticate. V-ați întrebat probabil, după parcurgerea capitolului 8, cum ar trebui să procedați pentru a realiza un formular care să actualizeze mai multe tabele (de exemplu, să permită adăugarea unei facturi, știind că aceasta constă într-o înregistrare în tabela FACTURI și *n* înregistrări în tabela LINIIFACT)[34] sau cum se creează un indicator de progres al unei operații care durează o perioadă mai mare de timp (import/export de date, proceduri de calcul generalizate).

Scopul acestui capitol este familiarizarea dezvoltatorului de aplicații cu utilizarea unor controale mai complexe, dar și mai puternice, în realizarea unor formulare care să ofere utilizatorului pe de o parte o funcționalitate sporită, iar pe de altă parte, ușurință și plăcere în exploatare. Pentru realizarea acestui deziderat, vom îmbina tabele derivate (*view*-uri) cu obiecte tip `Grid`, `OleBoundControl` (pentru câmpuri de tip general), `OleControl` etc. Ne vom referi și la modalitatea de tratare a erorilor ce pot apărea la execuția unui formular (interceptând evenimentul `Error`), în scopul de a evita blocarea aplicației și de a furniza utilizatorului un mesaj inteligibil în locul celui implicit.

Tot aici vom prezenta și modul de rezolvare a unei probleme destul de stânjenitoare, frecvent întâlnită în practică: sincronizarea obiectelor de tip `Combo-Box` cu surse de date de tip numeric.

10.1. Controlul *Grid* și controale pentru câmpuri de tip *General*

Fără îndoială, unul dintre cele mai complexe obiecte pentru manipularea datelor este `Grid`-ul. Prin intermediul său putem construi o interfață intuitivă, care, de obicei, satisface cu succes exigențele oricărui utilizator, furnizând următoarele avantaje:

- ușurință în deplasarea de la o înregistrare la alta a tabelei-sursă de date;
- viziune de ansamblu asupra unui număr mare de înregistrări simultan;
- ușurință în deplasarea de la un câmp la altul și în editarea acestora;
- posibilitatea încorporării în căsuțele `grid`-ului a altor tipuri de controale (gen `Combo-Box`, `CheckBox` etc.), în scopul restricționării introducerii datelor.

Controlul `Grid` este de fapt un obiect de tip container ce afișează datele, preluate de la sursa de date, sub formă de linii și coloane, într-un mod similar ferestrei `Browse`. Fiind un obiect de tip container, el *conține obiecte de tip coloană* (`Column`). La rândul ei, o coloană conține un obiect de tip `Header` și alte tipuri de controale (implicit `TextBox`, dar se pot plasa și altele). La execuție, aceste controale se pot multiplica în funcție de numărul de linii

34. În lucrările de specialitate, astfel de formulare se numesc *Master-Detail* sau *One-To-Many*.

al sursei de date sau, dimpotrivă, se pot afișa doar pe linia curentă. Pentru că atât `grid`-ul, cât și coloanele, `header`-ele și celelalte controale încorporate în coloane prezintă fiecare un set propriu de metode, evenimente și proprietăți, posibilitățile de control asupra `Grid`-ului sunt extrem de vaste.

Pentru câmpuri de tip `General` (care de obicei conțin imagini), în Visual FoxPro există un obiect specific: `OleBoundControl`.

Acestea sunt cele două controale noi pe care le vom utiliza în realizarea formularului de actualizare a tabelei PRODUSE (structura tabelei o regăsiți în capitolul 2). Figura 10.1 prezintă formularul despre care discutăm, aflat în faza finală a proiectării.

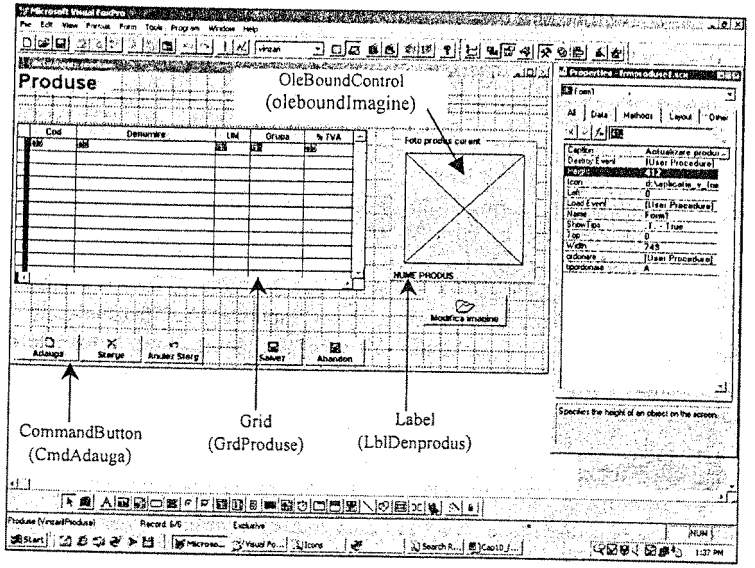


Figura 10.1. Macheta formularului de actualizare a datelor despre produse

Notă. Textele din paranteze arată numele obiectelor mai importante.

Înainte de a prezenta obiectele și proprietățile lor, să analizăm câteva detalii cu privire la modul în care va funcționa formularul la execuție:

- deși arată altfel, butoanele de comandă sunt același tip de obiect, `Command Button`, doar că afișează deasupra textului (proprietatea `Caption`) și o imagine a cărei sursă (un fișier `.bmp`) o regăsim în proprietatea `Picture`;
- coloanele `UM` și `GRUPA` din controlul de tip `Grid` conțin în plus un control de tip listă (`Combo-Box`), pentru a facilita introducerea unor valori dintr-o listă predefinită pe câmpurile respective;
- deplasarea de la o linie la alta în `Grid` asigură automat poziționarea pe înregistrarea corespondentă în tabelă; aceasta (PRODUSE) este precizată ca valoare a proprietății `RecordSource`, iar proprietatea `RecordSourceType` are valoarea `1 - Alias`;

- tot în momentul deplasării de la o înregistrare la alta, obiectele `OleBoundImagine` şi `LblDenprodus` se actualizează, afişând imaginea/numele produsului curent (corespunzător liniei din controlul `Grid` pe care este poziţionat cursorul);
- la un click pe titlul fiecărei coloane (`header`-ul acesteia), înregistrările se vor ordona după valorile coloanei respective, o dată ascendent, iar a doua oară descendent, în funcţie de starea proprietăţii `TipOrdonare` adăugate la formular (după procedeul explicat în capitolul 8);
- toate butoanele au asociată o combinaţie de taste pentru apelare rapidă (vezi în listingul 10.3 procedura evenimentului `KeyPress` la nivelul formularului), astfel:
  - `cmdAdauga`: Alt+Z;
  - `cmdSterge`: Alt+X;
  - `cmdAnulezSterg`: Alt+N;
  - `cmdSave`: Alt+S;
  - `cmdAbandon`: Alt+A.
- în momentul lansării formularului în execuţie, se deschide o tranzacţie (în sensul prezentat la capitolul 8) care se va comite sau anula în funcţie de opţiunea utilizatorului. La fel, după confirmarea (`cmdSave`) sau abandonul (`cmdAbandon`) unei tranzacţii, se declanşează una nouă. Aşadar, o tranzacţie nu mai corespunde actualizării unei înregistrări anume, ci corespunde unei perioade de timp (din momentul deschiderii formularului sau al apăsării unuia dintre butoanele `cmdSave` sau `cmdAbandon` şi până în momentul închiderii formularului sau, din nou, al apăsării unuia dintre cele două butoane);
- o procedură specifică de validare va verifica datele introduse pe coloana `Procent Tva`, astfel încât acestea să fie 0 sau 0.19.

Nu putem continua fără să lămurim un apect foarte important legat de controlul `Grid`: *adăugarea altui control decât cel implicit* (`TextBox`) *unei coloane*. Mai concret, adăugarea `Combo-Box`-ului în coloana `Grupa` presupune parcurgerea, în ordine, a următorilor paşi (vezi şi figura 10.2):

- selectăm obiectul de tip `Grid`, efectuăm click-dreapta şi alegem din meniul contextual opţiunea `Edit`, pentru a avea acces la obiectele componente (modalitate specifică de altfel oricărui container). Controlul `grdProduse` va căpăta un chenar gros, haşurat;
- ne poziţionăm pe coloana în care dorim să adăugăm controlul, fie pur şi simplu cu un click pe ea, fie căutând-o în fereastra `Properties`;
- selectăm butonul corespunzător din bara de controale (în cazul nostru `Combo-Box`) şi îl trasăm exact pe coloana selectată anterior (nu se acordă importanţă dimensiunii). Nu vom observa nici o modificare a `Grid`-ului, dar în fereastra de proprietăţi este vizibil (şi selectat) obiectul `Combo-Box`;
- selectăm din nou coloana în întregime şi, la proprietatea `CurrentControl` observăm (dacă totul a decurs conform planului) că putem opta pentru `Text1` sau `Combo1`. Vom selecta obiectul `Combo1`. Abia acum observăm o schimbare în controlul `grdProduse`, semnificând faptul că, de acum înainte, obiectul activ ce se va lega la sursa de date a coloanei va fi noul obiect de tip listă;

- atribuim proprietăţii `Style` a `Combo-Box`-ului valoarea `2-DropDownList`, în caz contrar elementele din listă fiind trunchiate ca lăţime (în momentul execuţiei), în funcţie de mărimea câmpului de care e legată coloana;
- proprietatea `Sparse` a coloanei va primi valoarea `.F.(False)`, astfel încât controlul de tip listă va fi vizibil, la execuţie, pe toate liniile coloanei. Dacă această proprietate are valoarea `.T.(True)`, obiectul de tip listă va fi vizibil, la execuţia formularului, numai dacă utilizatorul se poziţionează pe coloana respectivă.

Atenţie! O dată inserat un control pe o coloană, nu mai poate fi şters; poate fi doar dezactivat.

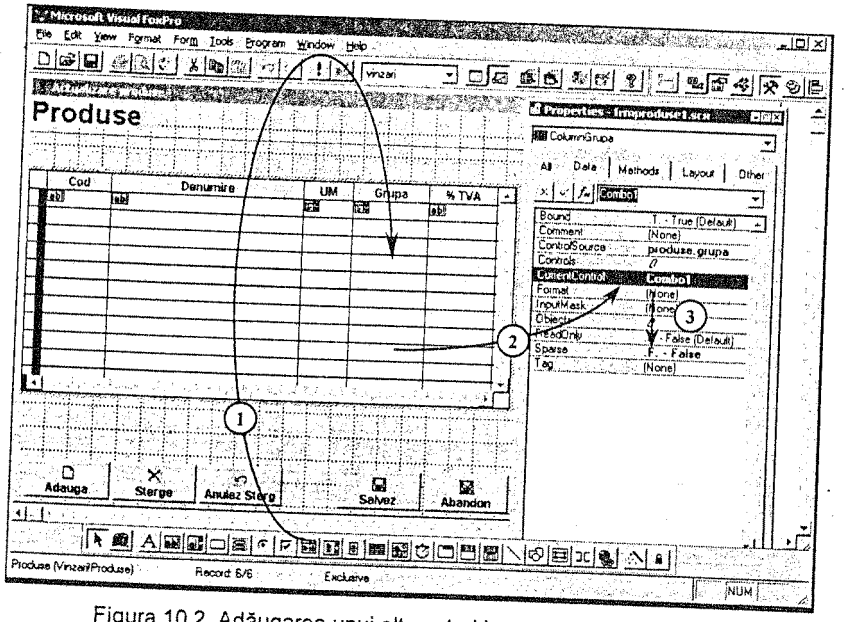


Figura 10.2. Adăugarea unui alt control la o coloană a `Grid`-ului

În tabelul 10.1 regăsim toate obiectele formularului (mai puţin cele de tip `Label`) configurate în funcţie de necesităţi. Numele obiectelor sunt, sperăm, suficient de semnificative pentru a le identifica şi în figura 10.1.

Tabelul 10.1. Obiectele formularului `frmProduse`

Proprietate	*Valoare*	*Comentarii*
	`Control: grdProduse`	
`RecordSource`	`Produse`	`Grid`-ul este legat la tabela `PRODUSE`.
`RecordSource Type`	`Alias`	Sursa de date este o tabelă (deschisă în prealabil).
`ColumnCount`	`5`	`Grid`-ul conţine 5 coloane.
	`Control: grdProduse.ColumnCodpr`	
`ControlSource`	`Produse.Codpr`	
	`Control: grdProduse.ColumnDenpr`	
`ControlSource`	`Produse.Denpr`	

Proprietate	*Valoare*	*Comentarii*
Control: grdProduse.ColumnUm		
ControlSource	Produse.Um	
CurentControl	Combo1	Controlul activ al coloanei este Combo1 (figura 10.2).
Sparse	.F.	Controlul activ va fi vizibil în momentul execuţiei pe toate liniile coloanei.
Control: grdProduse.ColumnGrupa		
ControlSource	Produse.Grupa	
CurentControl	Combo1	Vezi grdProduse.ColumnUm. Are acelaşi nume ca şi cel din coloana unităţii de măsură, dar poate fi schimbat după dorinţă.
Sparse	.F.	Vezi grdProduse.ColumnUm.
Control: grdProduse.ColumnProctva		
ControlSource	Produse. Proctva	
InputMask	9.99	Formatul de afişare a datelor-sursă – numeric cu un întreg şi două zecimale (separatorul zecimal este punctul).
Control: grdProduse.ColumnUm.Combo1		
ControlSource	Produse.Um	
RowSource Type	1- Value	Sursa de populare a elementelor din listă o vor constitui valorile introduse efectiv în proprietatea RowSource în faza de proiectare a formularului.
RowSource	Buc, Kg, L, M	Valorile ce vor constitui elementele din listă. Se introduc cu virgulă.
Style	DropdownList	
BorderStyle	0-None	Elimină aspectul 3D.
Control: grdProduse.ColumnGrupa.Combo1		
ControlSource	Produse.Grupa	
RowSource Type	1- Value	Vezi grdProduse.ColumnUm.Combo1.
RowSource	Tigari,Bere,Dulciuri,Rechizite,Carti, Altele	
Style	2-DropdownList	
BorderStyle	0-None	
Control:oleBoundImagine		
ControlSource	Produse. Imágine	
Stretch	2-Stretch	Imaginea-sursă se va redimensiona în funcţie de dimensiunile controlului.
Control:cmdAdauga		
Caption	Adauga	
Picture	="New.bmp"	Specifică fişierul de tip grafic ce conţine imaginea ce va fi afişată de obiect. Este precedat de semnul „=" şi inclus între ghilimele, pentru ca Visual FoxPro să-l caute în directoarele specificate prin SET DEFAULT şi SET PATH. Dacă fişierul nu este găsit, va fi generat un mesaj de eroare. Putem folosi şi adresa absolută ori relativă a fişierului pe disc (ex.: K:\aplicatie\imagini\butoane\New.bmp), în acest caz fără să mai folosim „=" şi ghilimelele. Imaginea trebuie să aibă, bineînţeles, dimensiuni adecvate.
MouseIcon	="hand-1.cur"	Specifică pictograma cursorului mouse-ului, la momentul execuţiei, când acesta se va afla deasupra obiectului[35].
MousePointer	99 -Custom	Valoare necesară pentru a personaliza pictograma mouse-ului (vezi MouseIcon).
ToolTipText	Adauga un produs nou	Scurt mesaj ce apare în momentul execuţiei, când poziţionăm mouse-ul deasupra unui obiect, cu scopul de a lămuri pe scurt funcţionalitatea acestuia. Pentru a funcţiona, este obligatorie setarea proprietăţii Form.ShowTips pe valoarea .T.(True).
Control:cmdSterge		
Caption	Sterge	
Picture	="Delete.bmp"	
MouseIcon	="hand-1.cur"	
MousePointer	99 -Custom	
ToolTipText	Sterge Produsul curent	
Control:cmdModifica,cmdSave,cmdAbandon,cmdImagine		
Au proprietăţi similare butoanelor prezentate anterior		
Formularul: FrmProduse		
Icon	="apps stop.ico "	Specifică pictograma din titlul ferestrei (vezi şi cmdAdauga).
Asters	.F.	Proprietate adăugată la nivel de formular. Ia valoarea .T.(True) dacă utilizatorul a marcat pentru ştergere cel puţin un produs.
TipOrdonare	A	Proprietate adăugată la nivel de formular ce specifică tipul de ordonare. Poate lua valoarea *A* sau *D* (ascendent/descendent).
KeyPreview	.F.-False	*Obligatorie pentru ca evenimentul KeyPress* al formularului să se declanşeze înainte de acelaşi eveniment al obiectelor componente.
WindowType	1-Modal	Blochează accesul la meniu sau alte formulare în timpul execuţiei.

35. Puteţi găsi fişiere utile pentru imagini, pictograme sau cursoare dacă veţi efectua o căutare pe disc (folosind utilitarul sistemului de operare) după toate fişierele cu extensii specifice (*.bmp, *.ico etc.).

Urmează să prezentăm codul metodelor şi evenimentelor obiectelor.

Listing 10.1. Metodele `Load` şi `Destroy` ale formularului `frmProduse` – varianta 1

Obiect: frmProduse(formularul)

```
*Eveniment: Load
IF NOT DBUSED('vinzari')
    OPEN DATABASE "\aplicatie\database\vinzari" SHARED
ENDIF
IF !USED('produse')
    USE produse IN 0
ENDIF
SELECT produse
SET SAFETY OFF          &&suprima avertismentul la crearea/stergerea fisierelor
INDEX ON denpr TAG denpr
INDEX ON um TAG um
INDEX ON grupa TAG grupa
INDEX ON proctva TAG proctva

BEGIN TRANSACTION               && se deschide o tranzacţie

*Eveniment: Destroy
IF TXNLEVEL()>0         && verifică nivelul tranzacţiei (în cazul nostru va fi intotdeauna 1)
    IF MESSAGEBOX('Salvati ultimele modificari?', 32+4,'Ultimele modificari nu au fost validate')=6
        END TRANSACTION
    ELSE
        ROLLBACK
    ENDIF
ENDIF
IF THISFORM.Asters=.T.          && vezi cmdSterge.click
    SELECT produse
    PACK
ENDIF

*vom sterge indecsii suplimentari  utilizati:
DELETE TAG denpr
DELETE TAG um
DELETE TAG proctva
DELETE TAG grupa

SET SAFETY ON           &&restabileste avertismentul la crearea/stergerea fisierelor
```

Notă. Indexarea tabelei PRODUSE după fiecare coloană este necesară pentru a putea ordona tabela după diverse criterii (cod, denumire, unitate de măsură), în funcţie de dorinţa utilizatorului. Trebuie să spunem însă că nu putem recurge la această metodă în cazul tabelelor deschise în mod partajat (lucru în reţea), deoarece comanda `INDEX ON` necesită acces exclusiv la tabelă. De asemenea, pentru tabelele cu multe înregistrări, procedura va fi destul de lentă.

Mai nimerit ar fi să construim de la bun început cei patru indecşi conform unor criterii de ordonare plauzibile, iar în procedura de ordonare să activăm indexul corespunzător prin comanda `SET ORDER`. Într-un asemenea caz, procedurile evenimentelor `Load` şi `Destroy` ar fi simplificate.

Atenţie! Utilizarea comenzii `PACK` (ştergere definitivă) în timpul execuţiei formularului duce la ***pierderea legăturii între obiecte şi sursa de date*** (`ControlSource` sau `RowSource`). Explicaţia decurge din comportamentul intern

al comenzii: închide tabela, şterge fizic datele şi apoi o redeschide. Pentru a restabili legătura, putem recurge la următorul artificiu:

```
Obiect.ControlSource=Obiect.ControlSource   şi/sau
Obiect.RowSource=Obiect.RowSource
```

Listing 10.2. Metodele `Load` şi `Destroy` ale formularului `frmProduse` – varianta 2

Obiect: frmProduse (formularul)

```
*Eveniment: Load
IF NOT DBUSED('vinzari')
    OPEN DATABASE "\aplicatie\database\vinzari" SHARED
ENDIF
IF !USED('produse')
    USE produse IN 0 SHARED
ENDIF
SELECT produse

BEGIN TRANSACTION               && se deschide o tranzacţie

*Eveniment: Destroy
IF TXNLEVEL()>0         && verifică nivelul tranzacţiei (în cazul nostru va fi intotdeauna 1)
    IF MESSAGEBOX('Salvati ultimele modificari?', 32+4,'Ultimele modificari nu au fost validate')=6
        END TRANSACTION
    ELSE
        ROLLBACK
    ENDIF
ENDIF
** stergerea definitiva nu poate fi realizatat pentru tabele deschise in mod partajat
** si va fi efectuata la inchiderea aplicatiei (vezi capitolul 17, listing 17.4)
```

Listing 10.3. Celelalte metode ale formularului `frmProduse`

Obiect: frmProduse(formularul)

```
*Eveniment: KeyPress
LPARAMETERS nKeyCode, nShiftAltCtrl
do case
    case nKeyCode=44            && ALT+Z
        thisform.cmdAdauga.click
    case nKeyCode=45            && ALT+X
        thisform.cmdSterge.click
    case nKeyCode=49            && ALT+N
        thisform.cmdAnulezSterg.click
    case nKeyCode=31            && ALT+S
        thisform.cmdSave.click
    case nKeyCode=30            && ALT+A
        thisform.cmdAbandon.click
endcase
** pentru codurile combinaţiilor de taste, vezi funcţia INKEY(), în documentaţia Visual FoxPro


*Metoda: Ordonare                       && adăugată de noi (vezi cap. 8 pentru lămuriri)
LPARAMETERS p_numeindex                 && parametrii se specifică aici cu LParameters
**Pentru modul de apelare vezi grdProduse.ColumnCodpr.Header1.Click
SELECT produse
IF THISFORM.TipOrdonare='A'
    SET ORDER TO &p_numeindex ASCENDING      && '&'-macrosubstituţie (cap. 4)
    THISFORM.TipOrdonare='D'
ELSE
```

```
    SET ORDER TO &p  numeindex DESCENDING
    THISFORM.TipOrdonare='A'
ENDIF

GO TOP
THISFORM.REFRESH
```

Obiect:grdProduse

```
Eveniment: AfterRowColChange   && se declanşează la deplasarea de la o celulă la alta,
                               && după ce noua celulă a primit controlul
LPARAMETERS nColIndex

THISFORM.lblDenprodus.CAPTION=Produse.Denpr
THISFORM.REFRESH
```

Obiect:grdProduse.ColumnCodpr.Header1

```
Eveniment: Click
THISFORM.Ordonare("Codpr")
```

Obiect:grdProduse.ColumnDenpr.Header1

```
Eveniment: Click
THISFORM.Ordonare("Denpr")
```

Obiect:grdProduse.ColumnUm.Header1

```
Eveniment: Click
THISFORM.Ordonare("Um")
```

Obiect:grdProduse.ColumnGrupă.Header1

```
Eveniment: Click
THISFORM.Ordonare("Grupa")
```

Obiect:grdProduse.ColumnProctva.Text1

```
Eveniment: Valid
**Text1 este controlul curent (implicit) al coloanei ColumnProctva
IF  NOT INLIST(THIS.VALUE,0,0.19)
    MESSAGEBOX( ' Nu puteti introduce decat valoarea 0 sau 0.19 ',16,' Valoare invalida ' )
    RETURN .F.
ENDIF
```

Obiect:cmdAdauga

```
Eveniment: Click
SELECT MAX(codpr)+1 FROM produse INTO ARRAY default_cod

INSERT INTO Produse (codpr,denpr,um,grupa,proctva)
            VALUES(default_cod,'Produs_default','L','Bere',0.19)

** nu am inclus in fraza INSERT câmpul general: Imagine

THISFORM.grdProduse.ColumnCodpr.SETFOCUS      && necesar pentru "cursivitatea"
actualizarilor
```

Obiect:cmdSterge

```
Eveniment: Click
SELECT Produse
DELETE
THISFORM.Asters=.T.
```

Obiect:cmdAnulezsterg


```
Eveniment: Click
SELECT produse
RECALL
```

Obiect:cmdSave

```
Eveniment: Click
END TRANSACTION                  && Comite tranzactia curenta
BEGIN TRANSACTION                && Deschide alta tranzactie
```

Obiect:cmdAbandon

```
Eveniment: Click
ROLLBACK                         && ANULEAZA tranzactia curenta (toate actualizarile
                                 && de la ulitma salavare sau de la ultimul abandon
SELECT produse
GO TOP                  && important pentru a nu lasa controlul pe o inregistrare care s-ar putea
                        && sa nu mai existe dupa ROLLBACK
THISFORM.REFRESH
BEGIN TRANSACTION
THISFORM.grdProduse.ColumnCodpr.SETFOCUS   && pentru a lăsa controlul pe gid
```

Obiect: cmdImagine

```
*Eveniment: Click && modifică imaginea produsului memorată în baza de date
vfis_nume=GETFILE("Bmp:bmp; Gif:gif; Img:tif ", "Nume fisier:", "OK", 0, ;
                  "SELECTATI FISIERUL DORIT")
IF ! EMPTY(vfis_nume)
    SELECT Produse
    APPEND GENERAL Imagine FROM &vfis_nume     && "&" - macrosubstitutie
ENDIF
THISFORM.REFRESH

** Pentru parametrii funcţiei GETFILE( ) vezi documentatia VFP
** functia GETFILE() se poate inlocui cu GETPICT()
```

În figura 10.3, care prezintă formularul în faza de execuţie, observăm următoarele:

- fiecare control de tip listă, în coloanele `UM` şi `Grupa` ale `grid`-ului, afişează valoarea de pe linia corespunzătoare;
- denumirea produsului curent se regăseşte sub imaginea sa;
- datele sunt ordonate după denumire în urma executării unui click pe titlul coloanei `Denumire`;
- controlul `OleBoundImagine` afişează imaginea produsului curent din controlul *grid*;
- butonul `cmdImagine` (inscripţionat *Modifică Imagine*) prezintă un alt tip de iconiţă a mouse-ului şi un scurt mesaj (`TooltipText`);
- la apăsarea butonului `cmdImagine`, se deschide fereastra de dialog pentru căutarea unui fişier pe disc (folosim funcţia `GETFILE()` – vezi listingul 10.3). Fereastra de dialog filtrează numai fişierele ce corespund tipului ales.

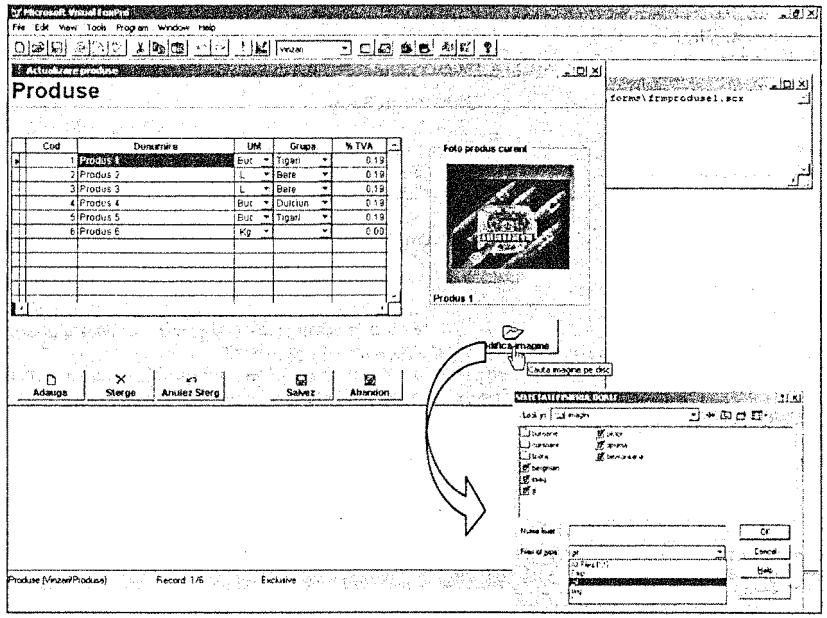


Figura 10.3. Formularul de actualizare a datelor despre produse, aflat în faza de execuție

10.2. Tratarea erorilor în faza de execuție a formularelor

Erorile ce pot apărea la execuția unui formular se împart în două mari categorii:

- erori *previzibile*, ce provin în general de la motorul bazei de date (încălcarea restricțiilor referențiale, a unicității cheii primare, a regulilor de validare, declanșatoarelor etc.);
- erori *imprevizibile*, ce pot avea diverse cauze legate, de cele mai multe ori, de inexistența pe disc a unui fișier necesar sau funcționarea defectuoasă a unor componente externe utilizate de aplicație etc.

Toate aceste tipuri de erori sunt tratate de obicei prin intermediul a două evenimente, `VALID` și `ERROR`, astfel:

- prin evenimentul `Error` al *formularului* putem trata în mod distinct *toate* erorile pe care ni le imaginăm posibile[36] și care pot proveni de la un obiect sau altul;
- prin evenimentul `Error` *al fiecărui obiect în parte* (anticipând erorile ce pot să apară în funcționarea obiectului respectiv);
- prin utilizarea evenimentului `Valid`, ce se declanșează înaintea evenimentului `Error`.

Ca metodă standard de lucru, cea mai indicată, zicem noi, este tratarea erorilor prin evenimentul `Error` al formularului. Rațiunea ar fi o modulare sporită a codului. Există

36. Pentru erorile „neimaginabile" vom prevedea întotdeauna o modalitate minimală de semnalare a lor.

totuși o problemă: deși în cazul altor obiecte, după declanșarea evenimentului `Error` la nivelul formularului, controlul este returnat obiectului ce a declanșat eroarea, *în cazul* `Grid`*-urilor este nevoie de invocarea explicită a metodei* `SetFocus` *pentru coloana care a declanșat eroarea.* Dacă nu procedăm în acest fel, riscăm blocarea formularului la apariția unei erori anume.

În cazul formularului exemplificat în paragrafele anterioare, metoda asociată evenimentului `Error` ar putea fi cea din listingul 10.3 bis.

Listing 10.3 bis. Tratarea erorilor în mod global la nivel de formular

```
                                    Obiect: FrmProduse
*Eveniment: Error
LPARAMETERS nError, cMethod, nLine
IF nError=1884                              && Uniqueness of index "name" is violated
   MESSAGEBOX('Ati utilizat deja acest cod de produs!!',16,'Se incalca restrictia de unicitate')
ELSE
   MESSAGEBOX("Numar:"+STR(nError)+",  ;
                          Metoda:"+cMethod+", Linia:"+STR(nLine) ,16, "Eroare neprevazuta")
ENDIF
THISFORM.REFRESH
THISFORM.grdproduse.SETFOCUS
```

Parametrii evenimentului `Error` sunt: numărul erorii, metoda care a declanșat-o și linia de cod cu pricina (vezi și tratarea erorilor prezentată în capitolul 4). Acești parametri *sunt transmiși implicit, la apariția unei erori, de obiectul care a declanșat-o.* Cu alte cuvinte, în formulare nu este nevoie de instrucțiunea `ON ERROR`.

Pentru o listă exhaustivă a numerelor (și mesajelor) de eroare, se va consulta documentația produsului Visual FoxPro (*help*-ul în format electronic, la rubrica `Reference→ ErrorMessages`). De asemenea, putem apela la funcțiile `ERROR(numar_eroare)`, care va genera eroarea specificată prin număr, precum și la funcția `MESSAGE()`, care va returna ultimul mesaj de eroare apărut în sesiunea curentă de lucru.

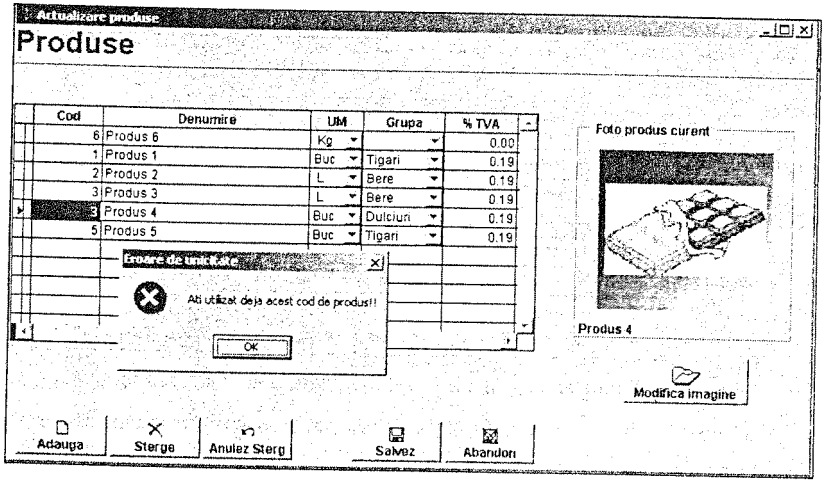


Figura 10.4. Apariția unei erori (tratate) la execuția formularului

Important! Prin tratarea erorii, dacă aceasta provine de la motorul bazei de date, se asigură şi restaurarea automată (*auto-revert*) a situaţiei iniţiale.

Figura 10.4 prezintă tratarea încălcării restricţiei de unicitate în momentul în care încercăm să introducem un cod deja existent (*3*) pentru „*Produs 4*".

10.3. Formulare şi tabele derivate. Specificul formularelor *Master-Detail*

Formularele descrise până acum aveau drept scop actualizarea unei singure tabele. Chiar dacă erau deschise mai multe tabele, unele dintre ele se foloseau exclusiv în scopul consultării, prin popularea unor controale tip listă.

Vom încerca, în paragraful ce urmează, să desluşim specificul formularelor „părinte-copil" (*Master-Detail* sau *One-To-Many*). Prin aceste sintagme înţelegem acele formulare care au drept scop actualizarea a două sau mai multe tabele simultan, tabele care sunt „relaţionate" (legate prin valorile cheilor primare şi străine) şi numai una este tabela-părinte, celelalte fiind tabele-copil. De regulă, unei înregistrări din tabela-părinte îi corespund mai multe înregistrări din tabela-copil, astfel încât numărul intervenţiilor pe cele două tabele este, aproape întotdeauna, diferit.

Ideea centrală ce trebuie urmărită în conceperea unui asemenea formular constă în faptul că acesta trebuie să asigure, pe lângă actualizarea datelor, şi o sincronizare a afişării acestora. Astfel, fiind poziţionat pe o linie anume din tabela-părinte, utilizatorul trebuie să poată accesa numai *înregistrările corespunzătoare din tabelele-copil.*

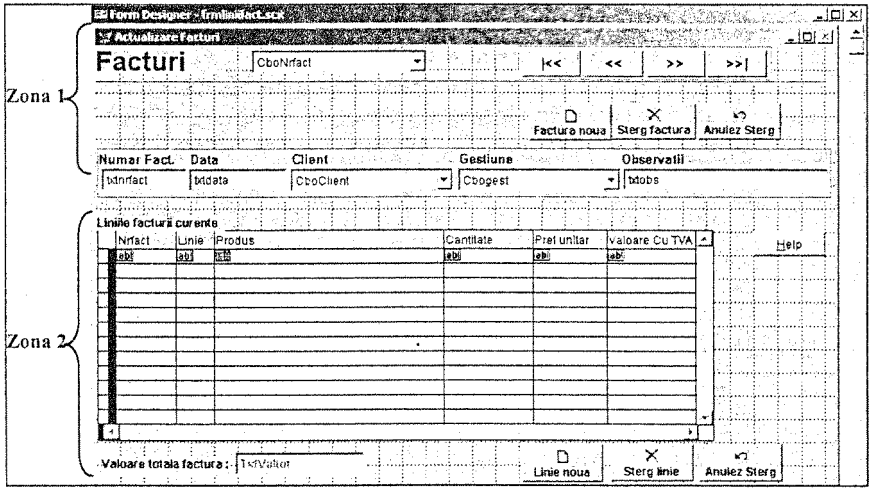


Figura 10.5. Formular de tip `One-to-Many` pentru actualizarea facturilor

Exemplul nostru va demonstra modul de realizare a unui formular de preluare a facturilor (figura 10.5), cu precizarea că nu urmărim şi crearea unui mecanism de protecţie la ştergere sau modificare a facturilor introduse în alte luni decât cea curentă (deci nu ne vom conforma întocmai regulilor financiar-contabile referitoare la închiderea lunii). De

asemenea, pentru simplificare şi pentru a ne concentra atenţia asupra aspectelor noi ce le aduce acest tip de formular, am eliminat tranzacţiile (nu există butoane de validare/anulare a actualizărilor). Firește, într-o etapă ulterioară (o dată cu sporirea experienţei dezvoltatorului de aplicaţii), aceste mecanisme, deocamdată omise din raţiuni didactice, vor putea fi implementate.

Aşa cum am văzut în capitolul 2, o factură (ca document) este reprezentată în baza de date prin intermediul înregistrărilor a două tabele (rezultat al normalizării): FACTURI şi LINIIFACT. Ca urmare, formularul va trebui să afişeze şi să permită actualizarea înregistrărilor celor două tabele luând în considerare legătura dintre ele: unei linii din tabela-părinte FACTURI îi corespunde mai multe linii în tabela-copil LINIIFACT. În acest sens, aşa cum reiese şi din figură, formularul va fi împărţit (convenţional) în două zone distincte:

- *zona 1* – corespunde tabelei FACTURI[37] şi oferă posibilitatea navigării prin înregistrările acesteia şi, eventual, editarea uneia sau alteia. Tot aici regăsim şi butoanele de adăugare/ştergere a unei facturi, ştergerea implicând pe de o parte ştergerea liniei curente din FACTURI, iar pe de altă parte, ştergerea tuturor liniilor corespondente din tabela LINIIFACT;
- *zona 2* – corespunde tabelei LINIIFACT[38] şi va afişa la un moment dat doar liniile corespunzătoare facturii curent selectate în zona 1. Aici regăsim butoane de adăugare/ştergere a unei linii dintr-o factură, corespunzătoare unei singure linii din tabela LINIIFACT.

Realizarea acestui formular presupune rezolvarea a două probleme specifice, nediscutate în nici unul din exemplele de până acum:

- controlul de tip `Grid` din zona 2 trebuie să filtreze înregistrările tabelei LINIIFACT şi să le afişeze numai pe cele corespunzătoare facturii selectate în zona 1;
- câmpurile `Facturi.codcl` şi `Liniifact.codpr`, fiind de tip numeric, nu pot fi legate direct prin proprietatea `ControlSource` la obiectele de tip `ComboBox` corespunzătoare din formular.

O modalitate clasică de rezolvare a primei probleme constă în construirea unei tabele derivate (`view`) parametrizate (vezi şi capitolul 9). Respectiva tabelă derivată va extrage la un moment dat doar acele înregistrări din tabela LINIIFACT ce au numărul de factură egal cu valoarea unei variabile globale. În timpul execuţiei formularului, această variabilă va fi actualizată permanent cu valoarea câmpului `Nrfact` de pe înregistrarea curentă din FACTURI. Ca urmare, prin reinterogare (funcţia `REQUERY()`), tabela derivată va conţine întotdeauna numai liniile aferente unei singure facturi.

Pentru cea de-a doua problemă există, de asemenea, un mod de rezolvare standard: vom renunţa la legarea directă prin `ControlSource` a celor două obiecte de tip listă; în schimb, vom utiliza metoda `Refresh` şi evenimentul `InteractiveChange` ale obiectelor respective pentru a asigura sincronizarea cu datele din tabele.

Secvenţa de cod pentru construirea tabelei derivate despre care vorbeam o regăsim în fragmentul următor:

---

[37] Vezi capitolul 6, paragraful 6.5.2, pentru coloana `Valtotala`.

[38] Vezi capitolul 6, paragraful 6.4, pentru coloana calculată `Valcutva` adăugată în tabela LINIIFACT.

Listing 10.4. Cod-sursă pentru crearea tabelei derivate `vliniifact`

```
*View pentru liniile unei facturi
nrfact_=1111 && variabila initializata cu o valoare aleatoare in scopul functionarii codului ce urmeaza
CREATE SQL VIEW vliniifact AS ;
    SELECT nrfact,linie,ALLTRIM(STR(codpr)) AS codpr1,cantitate,pretunit,valcutva FROM liniifact WHERE nrfact=nrfact_

DBSETPROP('vliniifact','view','tables','liniifact')

DBSETPROP('vliniifact.nrfact','field','keyfield',.T.)
DBSETPROP('vliniifact.linie','field','keyfield',.T.)

DBSETPROP('vliniifact.nrfact','field','updatename','liniifact.nrfact')
DBSETPROP('vliniifact.linie','field','updatename','liniifact.linie')
DBSETPROP('vliniifact.cantitate','field','updatename','liniifact.cantitate')
DBSETPROP('vliniifact.pretunit','field','updatename','liniifact.pretunit')

** adugarea regulilor de validare ce actualizeaza campul "Valcutva"
DBSETPROP('vliniifact.Cantitate', 'Field', 'RuleExpression','fvr_cant_pret_Vliniifact()')
DBSETPROP('vliniifact.PretUnit', 'Field', 'RuleExpression','fvr_cant_pret_Vliniifact()')

DBSETPROP('vliniifact.nrfact','field','updatable',.T.)
DBSETPROP('vliniifact.linie','field','updatable',.T.)
DBSETPROP('vliniifact.cantitate','field','updatable',.T.)
DBSETPROP('vliniifact.pretunit','field','updatable',.T.)
DBSETPROP('vliniifact.valcutva','field','updatable',.T.)

DBSETPROP('vliniifact', 'view', 'SendUpdates', .T.)
```

După cum observăm, faptul că atributul `Codpr` este de tip numeric complică puțin realizarea interfeței, prin simplul fapt că nu poate fi legat direct la un obiect de tip listă. Pentru că nu dorim să „chinuim" utilizatorul obligându-l să utilizeze direct codurile produselor (deși există magazineri care cunosc mai bine codurile decât denumirile produselor), vom recurge la un artificiu constând în următoarele:

- transformăm, în definiția tabelei derivate, câmpul `Codpr` din tip numeric în tip caracter (funcția `STR()` în combinație cu `ALLTRIM()`, pentru a elimina spațiile);
- câmpul nou-obținut, `Codpr1`, nu va fi legat direct la câmpul corespunzător (`Codpr`) din tabela-sursă LINIIFACT, așa încât orice actualizare a valorilor acestuia în tabela derivată nu va fi trimisă în tabela de bază;
- realizăm o legătură directă (proprietatea `ControlSource`) între obiectul de tip listă din coloana `ColCodpr` a `grid`-ului și câmpul corespunzător al tabelei derivate, `Codpr1`;
- orice modificare efectuată de utilizator pe această coloană o gestionăm prin intermediul evenimentului `InteractiveChange`, în care introducem câteva linii de cod ce vor realiza actualizările necesare direct în tabela de bază LINIIFACT, pe înregistrarea corespunzătoare.

Un alt aspect ce trebuie sesizat se referă la câmpul calculat `Valcutva`. Acesta este actualizat automat de procedura stocată `fvr_cant_pret_Vliniifact()`[39] în

39. Vezi capitolul 6, paragraful 6.4.

tabela-sursă, LINIIFACT, la orice actualizare trimisă din tabela derivată. Totuși, pentru ca tabela derivată să afișeze valoarea reală a acestui câmp, ar trebui să reinterogăm după fiecare actualizare tabela-sursă (funcția `REQUERY()`). O modalitate de a evita această situație este crearea unei alte proceduri stocate care să calculeze automat câmpul în tabela derivată (listingul 10.4 bis).

Listing 10.4 bis. Procedură stocată pentru actualizarea câmpului `Vliniifact.Valcutva`

```
PROCEDURE fvr_cant_pret_Vliniifact
*****************************************
REPLACE VLINIIFACT.ValCuTVA WITH VLINIIFACT.Cantitate * VLINIIFACT.PretUnit * ;
    ( 1 + afla_tva(Val(VLINIIFACT.CodPr1)))
RETURN .T.
ENDPROC
```

În formular, coloana corespunzătoare din obiectul de tip `grid` va trebui să fie `Read-Only`.

În continuare vom prezenta, în același mod cu care v-ați obișnuit deja din paragrafele anterioare, proprietățile și metodele obiectelor de pe formular.

Tabelul 10.2. Obiectele formularului `frmLiniiFact`

Proprietate	*Valoare*	*Comentarii*
	`Control: cboNrfact`	
`ControlSource`	`(none)`	Nu este legat la o sursă de date. Servește doar pentru căutarea unei facturi.
`RowSource Type`	`3-SQLStatement`	
`RowSource`	`SELECT nrfact, datafact, IIF(DELETED(), " VA FI STEARSA", "OK"+SPACE(11)) FROM facturi ORDER BY nrfact INTO CURSOR crsfact`	Lista facturilor, cu data fiecăreia și o coloană ce va afișa `OK` dacă factura nu este marcată pentru ștergere și `VA FI STEARSA` în caz contrar.
`BoundColumn`	`1`	Proprietatea `Value` va stoca valoarea curentă de pe prima coloană a listei (`nrfact`).
`ColumnCount`	`3`	
`ColumnWiths`	`100,100,250`	
`Style`	`2-DropDownList`	
	`Control: txtNrfact`	
`ControlSource`	`Facturi.Nrfact`	
`SelectOnEntry`	`.t.`	
	`Control: txtData`	
`ControlSource`	`Facturi. Datafact`	
`SelectOnEntry`	`.t.`	
	`Control: cboClient`	
`ControlSource`	`(none)`	Nu este legat direct la `Facturi.Codcl`, pentru că acest câmp este de tip numeric.

Proprietate	*Valoare*	*Comentarii*
		Sincronizarea se face prin intermediul metodelor `Refresh` și `InteractiveChange`.
`RowSource Type`	`3-SQLStatement`	
`RowSource`	`SELECT dencl, codcl FROM clienti INTO cursor crsclienti ORDER BY dencl`	
`BoundColumn`	`2`	Proprietatea `Value` va stoca valoarea curentă de pe a doua coloană (`codcl`); va fi de tip caracter.
`ColumnCount`	`2`	
`ColumnWiths`	`250,100`	
`Style`	`2-DropDownList`	
`Control: cboGestiune`		
`ControlSource`	`Facturi. Gestiune`	
`RowSource Type`	`3-SQLStatement`	
`RowSource`	`SELECT den_gest, gestiune FROM gestiuni INTO CURSOR crsgestiuni`	
`BoundColumn`	`2`	
`ColumnCount`	`2`	
`ColumnWiths`	`250,100`	
`Style`	`2-DropDownList`	
`Control: txtValtot (valoarea totala a facturii curente)`		
`ControlSource`	`Facturi.Valtotala`	Câmpul `Valtotala` este actualizat prin trigger-ul de update al tabelei `LINIIFACT` (vezi capitolul 6).
`Enabled`	`.F.-False`	
`InputMask`	`999,999,999.99`	Formatul de afișare a rezultatului.
`Control: grdVliniifact (grid-ul de pe formular)`		
`RecordSource`	`Vliniifact`	Tabela derivată construită anterior (vezi listingul 10.4).
`RecordSourceType`	`1-Alias`	
`Control: grdVliniifact.ColNrfact`		
`ControlSource`	`Vliniifact. Nrfact`	
`ReadOnly`	`.T. - True`	Utilizatorul nu va putea modifica valorile pe această coloană (numărul de factură este același pentru toate liniile facturii). Această coloană poate să lipsească, dar a fost totuși introdusă pentru a urmări sincronizarea înregistrărilor din cele două tabele.

Proprietate	*Valoare*	*Comentarii*
`DynamicBack Color`	`IIF(RECNO()=curenta_,RGB(255,255,8),RGB(0,64,128))`	Linia curentă (activă) capătă o altă culoare decât celelalte. Variabila globală `curenta_` este inițializată în `grdVliniifact.Init` și actualizată la evenimentul `AfterRowColChange` al `Grid`-ului. Pentru funcțiile `RGB()` și `RECNO()` vezi documentația Visual FoxPro.
`DynamicFore Color`	`IIF(RECNO()=curenta_,0,RGB(255,255,255))`	Similară proprietății `DynamicBackColor`, doar că se referă la culoarea caracterelor.
`Control: grdVliniifact.ColLinie`		
`ControlSource`	`Vliniifact.Linie`	
`ReadOnly`	`.T. - True`	Vezi `ColProdus`.
`DynamicBack Color`	`Idem ColNrfact`	
`DynamicFore Color`	`Idem ColNrfact`	
`Control: grdVliniifact.ColCodpr`		
`ControlSource`	`Vliniifact. Codpr1`	În tabela derivată, numele câmpului este `Codpr1`.
`DynamicBack Color`	`Idem ColNrfact`	
`DynamicFore. Color`	`Idem ColNrfact`	
`CurentControl`	`Combo1`	Pentru schimbarea controlului activ al coloanei, vezi explicațiile din paragraful 10.1.
`Sparse`	`.F.`	
`Control: grdVliniifact.ColCantitate`		
`ControlSource`	`Vliniifact. Cantitate`	
`DynamicBack Color`	`Idem ColNrfact`	
`DynamicFore Color`	`Idem ColNrfact`	
`Control: grdVliniifact.ColPretunit`		
`ControlSource`	`Vliniifact.Pretunit`	
`DynamicBack Color`	`Idem ColNrfact`	
`DynamicFore Color`	`Idem ColNrfact`	
`Control: grdVliniifact.ColValcutva`		
`ControlSource`	`Vliniifact. Vlcutva`	
`ReadOnly`	`.T.`	Nu permite modificare.
`Control: grdVliniifact.ColCodpr.CboCodpr (controlul activ al coloanei pentru produs din Grid)`		
`ControlSource`	`Vliniifact.Codpr1`	
`RowSource`	`3-SQLStatement`	

Proprietate	*Valoare*	*Comentarii*
Type		
RowSource	SELECT denpr, ALLTRIM(STR(codpr)) AS codpr FROM produse ORDER BY denpr INTO CURSOR crsdenpr	Codul produsului suferă aceeaşi transformare (tip caracter) ca şi în definiţia tabelei derivate pentru a lega obiectul la respectivul câmp.
BoundColumn	2	
ColumnCount	1	La deschiderea listei se va afişa o singură coloană: cea cu numele produselor. Acest fapt nu ne împiedică să o legăm pe cea de-a doua la sursa de date Vliniifact.Codpr1.
Style	2-DropDownList	
BorderStyle	0-None	

În ceea ce priveşte metodele şi evenimentele, vom începe prin a le prezenta pe cele „sensibile", specifice sincronizării datelor cu obiectele corespondente de pe formular (este vorba de tabela derivată Vliniifact şi câmpurile Liniifact.Codpr şi Facturi.Codcl).

Listing 10.5. Principalele metode ale formularului frmFacturi

```
                    Obiect: frmFacturi (formular)
*Eveniment: Load
PUBLIC nrfact_      && parametrul tabelei derivate trebuie declarat Public pentru a
                    && putea fi apelat de metodele altor obiecte
IF !USED('facturi')
    USE facturi IN 0 EXCLUSIVE
ENDIF
SELECT facturi
GO TOP
nrfact_=Facturi.nrfact          && initializarea parametrului tabelei derivate este obligatorie
                                && înainte de a o deschide

IF !USED('vliniifact')
    USE vliniifact IN 0
ENDIF

REQUERY( 'vliniifact' )     && pentru funcţia REQUERY(), vezi capitolul 9

IF !USED('gestiuni')
    USE GESTIUNI IN 0
ENDIF

                    Obiect: cboNrfact
*Eveniment: InteractiveChange
nrfact_=VAL(THIS.VALUE)     && actualizăm parametrul tabelei derivate cu valoarea curentă
                            && selectată din listă transformată în tip numeric
x=SEEK(nrfact_,'facturi','nrfact')  && cautare si pozitionare in tabela facturi
REQUERY('vliniifact' )
THISFORM.REFRESH
```



```
                    Obiect: cboClient
*Metoda:Refresh && asiguram sincronizarea obiectului cu valoarea curentă Facturi.Codcl

codcl_=facturi.codcl
FOR i=1 TO THIS.LISTCOUNT
    IF VAL(THIS.LIST(i,2))=codcl_
        THIS.LISTINDEX=i
    ENDIF
ENDFOR
** THIS. – specifica obiectul curent (echivalent cu THISFORM.cboClient. ...)

*Eveniment:InteractiveChange  && asiguram sincronizarea valorii curente Facturi.Codcl cu
                              && noul element ales din listă
REPLACE facturi.codcl WITH VAL(THIS.VALUE)
** nu uităm că BoundColumn este 2

                    Obiect: grdVliniifact.ColCodpr.Combo1
*Eveniment:InteractiveChange  && asigura sincronizarea valorii curente Liniifact.Codpr cu
                              && noul cod al produsului selectat din listă
curentnrfact_=vliniifact.nrfact
curentlinie_=vliniifact.linie
UPDATE liniifact SET codpr=VAL(THIS.VALUE) ;
    WHERE nrfact=curentnrfact_ AND linie=curentlinie_
** nu uităm că noul câmp Codpr1, al tabelei derivate la care este legat controlul, nu trimite
actualizările în tabela de bază Liniifact

                    Obiect: grdVliniifact
*Eveniment:Valid  && absolut necesara în cazul Grid-urilor legate la tabele derivate
SELECT vliniifact
    TABLEUPDATE()   && asigura comiterea modificărilor liniei curente din
                    && Vliniifact în tabela de bază Liniifact , atunci când grid-ul pierde
                    && controlul

                    Obiect: txtNrfact
*Eveniment:LostFocus  && asigurăm sincronizarea obiectelor la modificarea unui număr de
                      && factură

nrfact_=THIS.VALUE
THISFORM.cbonrfact.REQUERY      && reincarca lista de elemente
THISFORM.cbonrfact.REFRESH

FOR i=1 TO THISFORM.cbonrfact.LISTCOUNT
    IF  ALLTRIM(THISFORM.cbonrfact.LISTITEM(i))==ALLTRIM(STR(facturi.nrfact))
        THISFORM.cbonrfact.LISTINDEX=i
    ENDIF
ENDFOR
REQUERY('vliniifact')           && reactualizează tabela derivată
THISFORM.REFRESH
```

În continuare vom prezenta şi celelalte metode şi evenimente ce completează „comportamentul" formularului.

Listing 10.6. Alte metode ale formularului frmFacturi

```
                    Obiect: grdVliniifact
*Eveniment: Init && vezi proprietăţile DynamicBackColor şi DynamicForeColor ale coloanelor
PUBLIC curenta_
curenta_=RECNO()
```

```
erRowColChange
()  &&variabila preia numarul inregistrarii curente din Vliniifact


tot.REFRESH

                    Obiect: txtNrfact
d
0
OX('Nr facturii trebuie sa fie > 0',16,'Eroare la introducere date' )


          Obiect: cmdAdaugaFactura (CommandButton)
k
dcl
stiune
cturi VALUES(0,DATE(),gest_,codcl_,'',0,{  /  /  },0,0)
u valori implicite pe cheile straine

act')
RESH
act.SETFOCUS
                    Obiect: cmdStergFactura
k
ai intai liniile copil din tabela  Liniifact si abia apoi linia curenta din tabela Facturi
ger-ul de stergere din tabela facturi (trebuie să nu ia în
niile-copil marcate pentru stergere)
iniifact WHERE nrfact=nrfact_


lrfact.REQUERY() && reactualizam lista

                    Obiect: cmdAnulezStergFactura
k



fact=nrfact_
rfact.REQUERY()

                    Obiect: cmdAdaugaLinie
k
M(STR(produse.codpr))  && va fi codul de produs implicit pentru o linie noua


ie) FROM vliniifact INTO ARRAY alinie
niifact VALUES (nrfact_,alinie(1)+1,produs_,0,0)
t
)
t.codpr WITH VAL(produs_)
rdvliniifact.SETFOCUS

                    Obiect: cmdStergLinie
k
t


                    Obiect: frmFacturi (formularul)
```



```
*Eveniment:Error
** tratarea erorilor la nivel de formular (global)
LPARAMETERS nError, cMethod, nLine
text_=""
do case
    case nerror=1585 && update conflict
        messagebox("Datele inregistrarii curente au fost modificate intre timp ",16,"Eroare
actualizare")
        tablerevert()
        requery()  && numai daca nu suntem intr-o tranzactie
        text_="thisform.grdvliniifact.setfocus"
    case nerror=1884 && uniqueness of index ....
        messagebox('Mai aveti odata acest numar de factura',16,'eroare unicitate')
        text_="thisform.txtnrfact.setfocus"
    otherwise
        messagebox("Nr: "+ str(nerror)+"metoda: "+cMethod,16,"Eroare neprevazuta")
endcase

if !empty("text_")
    &text_     && macrosubstitutie
endif
```

10.4. Formulare parametrizate

Este important de ştiut faptul că un formular poate fi apelat cu parametri şi poate returna valori.

Parametrii formularului vor fi preluaţi prin intermediul evenimentului `Init`, prima instrucţiune a acestuia fiind:

```
LPARAMETERS parametru1, ..., parametru n
```

Ştim că, atunci când o procedură parametrizată este lansată în execuţie fără parametri actuali, aceştia au întotdeauna valoarea logică `.F.(False)`. Ca urmare, în cazul în care formularul trebuie să funcţioneze şi independent de modulul apelant, va trebui să introducem un test suplimentar asupra tipului parametrilor cu ajutorul funcţiei `VARTYPE()`.

Parametrii actuali vor fi transmişi formularului prin comanda:

```
DO FORM <NumeFormular> WITH <p1>, ..., <pn> [TO <variabilă>]
```

Clauza `TO <variabilă>` a comenzii de mai sus specifică variabila ce va prelua (dacă este cazul) rezultatul returnat de formular. Pentru ca variabila să capete o valoare, e nevoie de o instrucţiune `RETURN <valoare>` *la sfârşitul evenimentului* `Unload` al formularului respectiv.

Atenţie! Dacă formularul returnează o valoare, este obligatoriu ca acesta să fie ***modal*** (`WindowType=1`). *În caz contrar, execuţia modulului apelant nu este suspendată.* Acest fapt poate genera diferite erori atât timp cât formularul nu a returnat o valoare.

Din păcate, se pare că nu putem prelua rezulatul unui formular decât printr-o singură variabilă. În cazul în care trebuie totuşi să fie returnate mai multe valori, limitarea poate fi depăşită prin „împachetarea" acestora într-un singur şir de caractere ce va conţine un caracter cu rol de separator:

```
p1="alfa"
```

```
p2="beta"
RETURN p1+"~"+p2
```

„Recuperarea" ulterioară a valorilor în programul apelant poate fi realizată cu ajutorul funcțiilor specifice `LEFT()`, `RIGHT()`, `SUBSTR()`, în combinație cu funcția `AT()`, menită să găsească pozițiile caracterelor de delimitare. Recunoaștem că procedeul este greoi, dar comanda `DO FORM ... TO ...` nu permite transmiterea parametrilor *prin referință*[40] „dinspre" formular spre modulul apelant. O altă soluție ar putea fi utilizarea unei variabile publice de tip tablou sau a unei tabele temporare cu o singură coloană, populată cu valorile de returnat, urmând a se transmite „afară" numele ei.

Mai trebuie să precizăm aici că putem apela proprietățile și metodele obiectelor unui formular aflat în execuție, dintr-un alt program ce se execută în paralel, prin intermediul unei proprietăți a variabilei sistem de tip obiect `_SCREEN`[41], astfel:

```
_SCREEN.ACTIVEFORM.Obiect.Metodă sau
_SCREEN.ACTIVEFORM.Obiect.Proprietate
```

10.5. Utilizarea controalelor ActiveX în formularele Visual FoxPro

Sistemul de operare Windows a oferit încă de la primele versiuni posibilitatea comunicației între diferite tipuri de aplicații, atât sub forma schimbului de date (de exemplu, operația de copiere a unui grafic din Excel în Word), cât și ca încorporare într-o aplicație a unui obiect specific altei aplicații, păstrându-i acestuia din urmă funcționalitățile native.

10.5.1. Ce este și ce-și propune tehnologia ActiveX ?

Tehnologia ActiveX se bazează pe folosirea într-o aplicație a unui obiect creat cu mijloacele unei explicații diferite. În acest caz, spunem că una dintre aplicații joacă rolul de *server* de obiecte, iar cealaltă rolul de *client*[42]. Sistemul de obiecte distribuite și protocolul corespunzător dezvoltate de Microsoft au format la început tehnologia *OLE* (*Object Linking and Embedding*). Ulterior a apărut tehnologia *ActiveX*, o perfecționare (cu deschidere către Internet) a tehnologiei OLE, care a fost la rândul ei rebotezată *Automation*.

După modul în care codul executabil al serverului interacționează cu cel al clientului, aplicațiile-server pot fi de două feluri:

- *in-process* – nu pot funcționa independent, ci numai dacă sunt apelate de o aplicație-client care încorporează codul executabil al serverului în propriul spațiu de adrese (pe acest principiu funcționează bibliotecile `.DLL` și `.OCX` ale Windows-ului). Este o soluție caracterizată prin viteză, dar vulnerabilă la defecțiunea oricăreia dintre cele două aplicații.
- *out-of-process* – serverul se execută în propriul spațiu de memorie, schimbul de date cu aplicația-client realizându-se prin diferite tehnici de comunicare interprocese. Aceste tehnici consumă mai multe tipuri de resurse și, ca urmare, vor determina timpi de execuție ceva mai mari. În contrapartidă, apare toleranța mai bună la erori: dacă serverul ori clientul „sucombă", cealaltă aplicație își păstrează totuși integritatea. Practic, orice aplicație Windows cu extensia `.EXE` ar putea servi ca server *out-of-process*, exemplele „clasice" fiind Excel, Visual Basic, clienții de e-mail etc.

40. Vezi capitolul 4, pentru modalități de transmitere a parametrilor între module.
41. Vezi documentația Visual FoxPro, capitolul 17 și capitolul 1, pentru alte proprietăți și metode ale variabilei sistem de tip obiect `_SCREEN`.
42. Atragem atenția că termenii *client* și *server* nu definesc aici două calculatoare diferite, ci două programe diferite, care se pot executa pe același sistem de calcul sau pe sisteme diferite.

Revenind la cazul nostru, putem spune că utilizarea obiectelor „străine" mediului Visual FoxPro prezintă în momentul de față două nuanțe. Pe de o parte, mecanismul `Automation` permite inserarea unui obiect al unei alte aplicații în următoarele locații:

- un câmp de tip `General` dintr-o tabelă (ați văzut deja cum se memorează fotografia unui produs, dar gândiți-vă că s-ar putea memora CV-ul unui angajat, scris în Word);
- un raport sau un formular (prin utilizarea controlului `ActiveX (OLE)`).

Pe de altă parte, anumite controale `ActiveX` (fișiere cu extensia `.OCX`), care nu țin de o aplicație anume, pot fi utilizate în formularele Visual FoxPro pentru îmbunătățirea interfeței, în caz că dezvoltatorul nu se mulțumește cu puterea celor 20 de controale native Visual FoxPro (din bara de instrumente `Form Controls`). În acest capitol ne vom ocupa numai de posibilitatea folosirii acestor noi controale și nu a programării (creării) lor.

Aplicația-client (Visual FoxPro) poate apela, prin intermediul obiectelor incluse, funcționalități ale aplicației-server (spre exemplu, pentru a actualiza un grafic Excel bazat pe o sursă de date FoxPro, se poate lansa din Excel o interogare `SQL` adresată mediului FoxPro). Obiectele `ActiveX` expun aplicației-client doar acele evenimente, proprietăți și metode strict necesare, fără să necesite din partea dezvoltatorului cunoștințe suplimentare referitoare la arhitectura internă a componentelor respective.

Aplicațiile Windows pot folosi o multitudine de componente `ActiveX` gata dezvoltate: controale `ActiveX` stocate în bibliotecile sistemului de operare Windows ori instalate pe sistemul de calcul o dată cu alte pachete software (spre exemplu, Visual Studio) sau procurate de la terți (livrate pe suporturi clasice sau descărcate de pe Internet). În plus, programatorii pot dezvolta propriile lor controale `ActiveX`, pentru a face față unor situații variate. Din acest punct de vedere, utilizatorii pachetului Microsoft Visual Studio se pot folosi de mediile Visual C++, Visual Basic (numai versiunile *Professional* și *Enterprise*) – ambele permit crearea de componente `.OCX`, `.EXE` și `.DLL` și chiar Visual FoxPro (numai pentru `.EXE` și `.DLL`).

10.5.2. Utilizarea controlului `Chart` într-un formular

Conform sintagmei „o imagine face cât o mie de cuvinte", există multe situații în care utilizatorii doresc să obțină dintr-o bază de date situații de ieșire ceva mai sugestive decât liste întregi de cifre. Pentru satisfacerea unor asemenea cerințe, dispunem de controlul `Microsoft Chart`, care a fost conceput ca o cale simplă de a afișa sub formă grafică serii de date[43].

Înainte de a prezenta un formular Visual FoxPro care folosește controlul `Chart`, vom exemplifica modul general de inserare a controalelor `ActiveX` într-un proiect Visual FoxPro. Se poate proceda în două moduri:

43. Există și o facilitate nativă Visual FoxPro de realizare a graficelor, dar flexibilitatea controlului `Chart` este net superioară.

1) Pe formular se trasează un control ActiveX (OLE), ca în figura 10.6, apoi, la apariţia dialogului din figura 10.7, se procedează la selectarea componentei dorite. Întrucât va fi afişată o listă cu toate controalele ActiveX disponibile pe sistemul respectiv de calcul, trebuie să cunoaştem, în această etapă, doar numele celui care ne interesează.

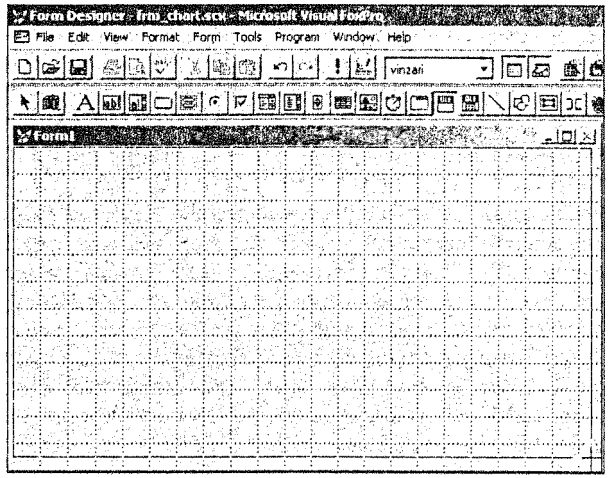


Figura 10.6. Trasarea unui control ActiveX (OLE)

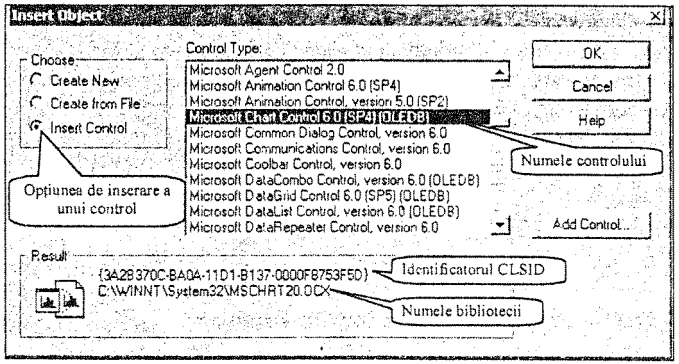


Figura 10.7. Dialogul de inserare a unui control (versiunea 1)

Atenţie! Pentru ca fereastra de dialog să arate ca în figura 10.7, trebuie instalat Visual Studio ServicePack 3 sau o versiune ulterioară.

În exemplul nostru, alegem Microsoft Chart Control 6.0 (numărul versiunii poate diferi, după configuraţia software a fiecărui sistem în parte). În partea de jos a

ferestrei regăsim atât identificatorul unic CLSID[44], cât şi calea şi numele fişierului-bibliotecă .OCX[45].

2) Selectăm opţiunea Tools→Options...→Controls a meniului principal. Fereastra afişată va fi cea din figura 10.8, unde se bifează controalele dorite. Printr-un click pe butonul View Classes din bara de instrumente Forms Designer, se alege opţiunea ActiveX (figura 10.9). Bara de instrumente Forms Designer îşi va schimba aspectul, afişând doar controalele adăugate la proiect prin opţiunea de meniu Tools→Options...→Controls (vezi figura 10.10).

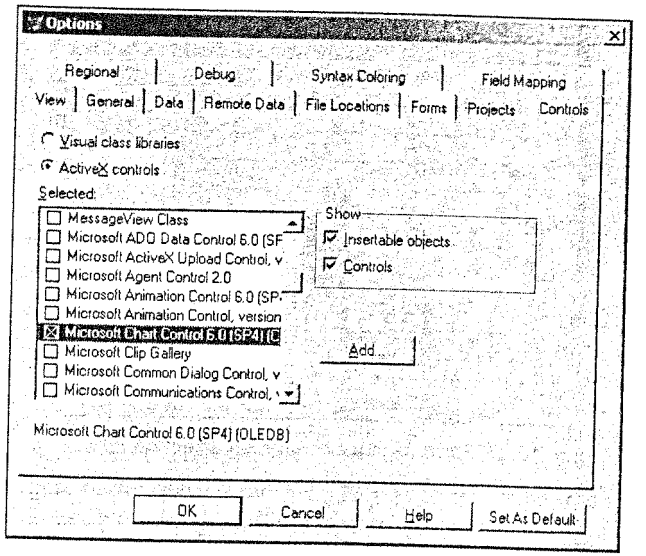


Figura 10.8. Dialogul de adăugare la proiect a controalelor ActiveX (versiunea 2)

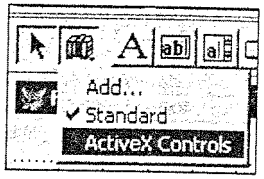


Figura 10.9. Opţiunea pentru bara de controale ActiveX

În acest ultim caz, plasarea controlului Chart pe formular are loc prin selecţia butonului indicat în figura 10.10.

44. Sistemul de operare Windows a fost proiectat după o arhitectură orientată-obiect. Pentru o funcţionare armonioasă, se atribuie fiecărui tip de obiect un identificator unic numit CLSID (*CLaSs IDentifier*). Acesta este memorat în baza de date Registry.
45. Există şi biblioteci care includ mai multe controale, dar Visual FoxPro are grijă să le disocieze şi să le prezinte individual în listă.

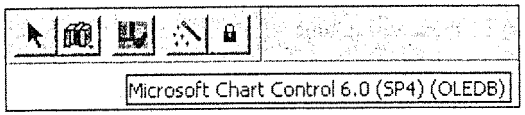


Figura 10.10. Linia de instrumente ActiveX, după alegerea controlului Chart

În ambele cazuri, rezultatul obținut arată ca în figura 10.11.

Notă. Pentru plasarea altor controale pe formular (din cele native VFP), e necesar să ne întoarcem la bara de instrumente Standard, prin meniul arătat în figura 10.9.

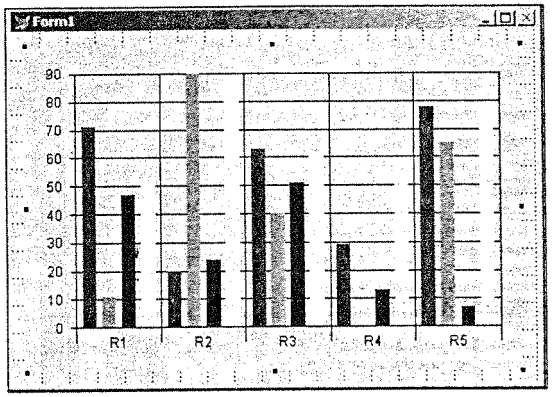


Figura 10.11. Formular care conține un obiect de tip Chart

Aspectul inițial al obiectului de tip Chart poate fi modificat, în limite largi, folosind proprietățile sale. Pentru început, vom da proprietății Name valoarea OleChart. Alte proprietăți notabile sunt redate în tabelul următor.

Tabelul 10.3. Principalele proprietăți ale controlului Chart

Proprietate	*Semnificație*
ChartType	Tipul de grafic (1-2d Bar înseamnă histogramă simplă, 13-3d Pie este diagramă de structură tridimensională etc.). Sunt disponibile într-o listă.
Column	Coloana curentă (proprietate utilă la popularea cu date).
ColumnCount	Numărul de serii de date (în cazul de față, numărul de coloane per categorie este 4).
ColumnLabel	Numele seriei de date (exemplu: cifra de afaceri, profit, salariu etc.).
Row	Linia curentă (proprietate utilă la popularea cu date).
RowCount	Numărul de categorii (în cazul de față, R1...R5).
RowLabel	Numele categoriei (exemplu: firma, an, nume salariat etc.).
DATA	Valoarea curentă (linia *n*, coloana *m*) din sursa de date.
ShowLegend	Afișează (.T.) sau nu (.F.) legenda.
TitleText	Titlul graficului.

De asemenea, multe proprietăți sunt accesibile prin pagina de proprietăți proprie controalelor ActiveX. Pagina se obține prin click-dreapta pe controlul respectiv, urmat de selecția opțiunii <nume-control> Properties din meniul contextual (figura 10.12).

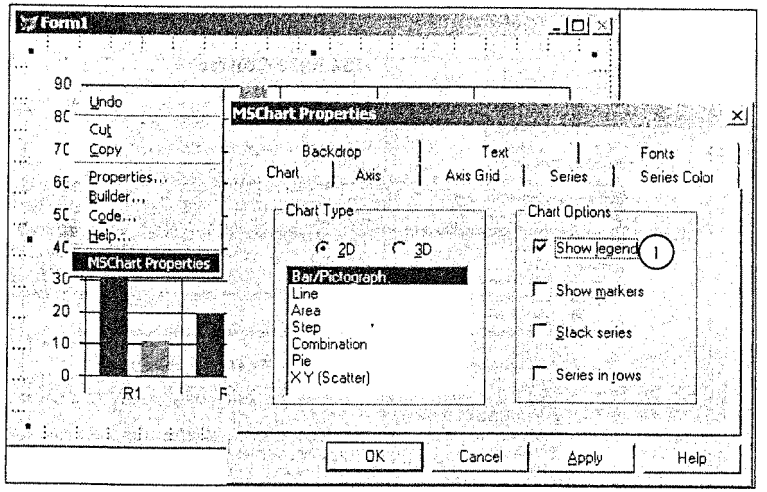


Figura 10.12. Fereastra de proprietăți specifică unui control ActiveX

După cum se observă, această fereastră are nu mai puțin de 8 cadre de pagină, fiecare cuprinzând proprietăți prezentate într-o formă mai accesibilă decât în tradiționala fereastră Properties a mediului Visual FoxPro. Pe pagina redată în figură se observă tipul de grafic selectat, precum și opțiunea pentru afișarea legendei. Și, fiindcă veni vorba de legendă, precizăm că poziția ei poate fi modificată. Din nefericire, atât poziția legendei, cât și mulți alți parametri sunt accesibili doar de manieră „programatică" (prin cod). Unele dintre proprietăți generează chiar erori dacă se încearcă modificarea valorilor lor direct în fereastra Properties.

Pentru o listă exhaustivă a proprietăților și metodelor controlului Chart vă recomandăm documentația MSDN în format electronic (vezi figura 10.13). Atenție, va trebui să consultați rubrica Visual Basic Documentation, care s-ar putea să nu fie activată dacă nu ați selectat opțiunea corespunzătoare la instalarea documentației MSDN.

Pentru exemplificarea modului de utilizare a acestui control, pornim de la ipoteza că am dori un grafic de tip histogramă care să reprezinte volumul vânzărilor cu și fără TVA, pentru anumite zile. La o primă vedere, ne sunt necesare două *serii de date* (se vor reprezenta ca bare a căror dimensiune se reprezintă pe ordonată și care au culori diferite): vânzările fără TVA, respectiv vânzările cu TVA. Cât despre *categorii* (care în graficele histogramă sunt puncte convenționale pe abscisă, fără legătură cu vreo unitate de măsură), acestea vor fi determinate de fiecare zi (dată calendaristică) pentru care avem vânzări. Prin urmare, pentru început vom atribui proprietății ColumnCount valoarea 2. Nu ne putem pronunța asupra valorii proprietății RowCount, întrucât nu știm, deocamdată, despre câte date calendaristice distincte este vorba...

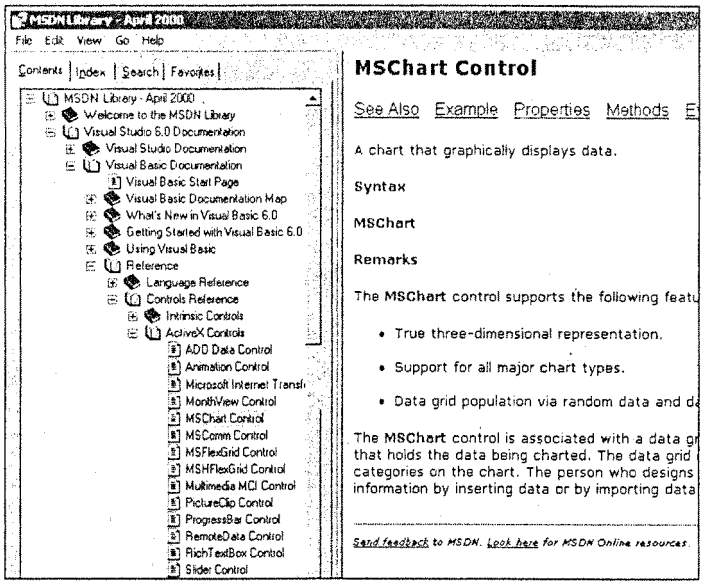


Figura 10.13. Instrucțiuni de folosire a controalelor `ActiveX` incluse în pachetul Visual Studio

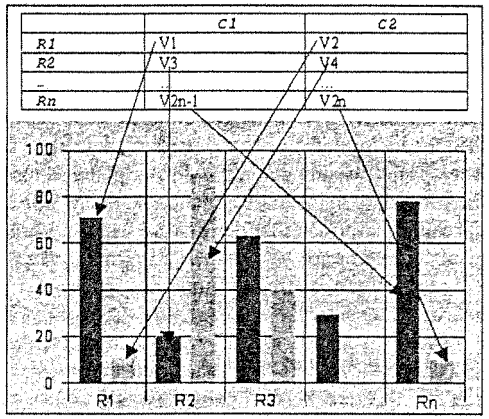


Figura 10.14. Corespondența valorilor din sursa de date cu coloanele graficului histogramă

Dacă sursa de date îmbracă o formă matriceală, corespondența relativ la coloanele histogramei este ilustrată în figura 10.14. Listingul 10.7 prezintă secvența de cod necesară stabilirii acestei corespondențe, plasată în evenimentul `Init` al formularului care găzduiește controlul `Chart`. Sursa de date este obținută printr-o interogare SQL care, zicem noi, se explică prin ea însăși.

Listing 10.7. O primă versiune a procedurii-eveniment `Init`

```
IF !DBUSED('vinzari')
    OPEN DATABASE '\aplicatie\database\vinzari' SHARED
ENDIF

SELECT datafact, SUM(cantitate*pretunit) AS Val_zi_fTVA, ;
    SUM(cantitate*pretunit*1.19) AS Val_zi_cuTVA ;
    FROM facturi F INNER JOIN liniifact lf ON F.nrfact=lf.nrfact ;
    INTO CURSOR sursa_grafic ;
    GROUP BY datafact ;
    ORDER BY datafact

SELECT sursa_grafic

THISFORM.oleChart.ColumnCount=2          && graficul va avea doua serii de date:
                                         && valoarea fara TVA si valoarea cu TVA
* graficul are atatea categorii cate linii are tabela-sursa:
THISFORM.oleChart.RowCount=RECCOUNT('sursa_grafic')
* populatea cu date folosind o bucla SCAN pe cursorul SURSA_GRAFIC
SCAN
    WITH THISFORM.oleChart
        .Row=RECNO() && pozitionare pe categoria care corespunde liniei curente din tabela-sursa
        .Column=1 && pozitionare pe coloana (seria de date) 1, linia curenta
        .Data=sursa_grafic.Val_zi_fTVA && atribuire sursa de date pentru coloana 1
        .Column=2 && pozitionare pe coloana (seria de date) 2, linia curenta
        .Data=sursa_grafic.Val_zi_cuTVA && atribuire sursa de date pentru coloana 2
        .RowLabel=ALLTRIM(DTOC(datafact))
    ENDWITH
ENDSCAN
```

La execuție, formularul va afișa graficul din figura 10.15. Deși aspectul acestuia este, deocamdată, mai mult decât discutabil, putem zice totuși că pasul cel mai greu a fost realizat: graficul reflectă valorile din sursa de date. Pentru a îmbunătăți forma graficului, urmează să-l completăm cu un titlu, o afișare mai inteligibilă a cifrelor de pe axa `Oy`, să suprimăm cea de-a doua axă `Oy` (din dreapta), să ajustăm aspectul axei `Ox` (datele calendaristice sunt scrise pe verticală, deoarece nu există suficient spațiu pe formular). De asemenea, este necesară o legendă, dar care să fie amplasată sub graficul propriu-zis, spre a câștiga spațiu pe orizontală (elementele graficului se redesenează proporțional, în funcție de dimensiunile – `Height` și `Width` – obiectului `Chart`). Cu alte cuvinte, parcurgem etapa numită „rafinarea graficului".

Pentru început, afișarea legendei în poziția implicită (dreapta-sus) are loc prin activarea opțiunii `Show Legend` (notată cu 1 în figura 10.12). Cea de-a doua axă `Oy` se suprimă urmând indicațiile din figura 10.16 (dezactivăm opțiunea `ShowScale` pentru axa `Second Y Axis`). Secvența de cod prezentată în listingul 10.8 rezolvă celelalte aspecte ale etapei de rafinare dacă va fi adăugată la procedura evenimentului `Init` (listingul 10.7).

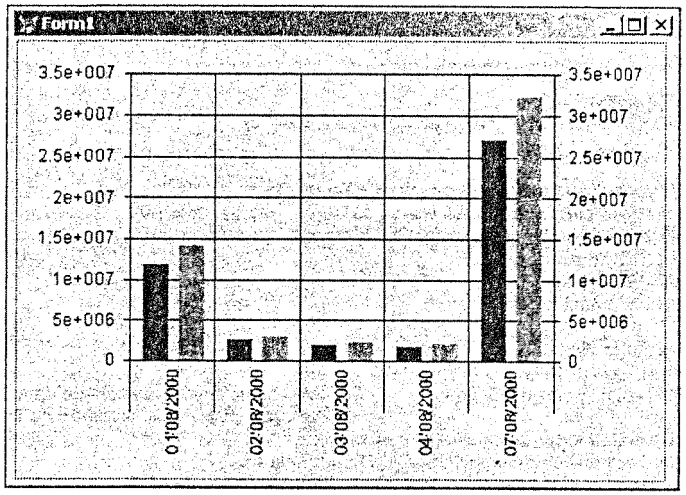


Figura 10.15. Graficul vânzărilor pe zile (varianta inițială)

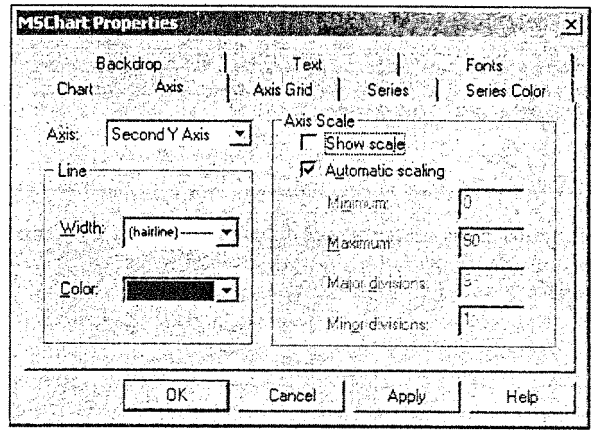


Figura 10.16. Opțiunea pentru axa secundară Oy

Listing 10.8. Completarea procedurii-eveniment `Init`

```
** continuare listing 10.7
* atribute vizuale ale graficului
WITH THISFORM.oleChart
    * se stabileste formatul etichetelor de pe axa Oy,
    * ca sa incapa milioanele
    FOR i=1 to .Plot.Axis(1).Labels.Count
        * atentie, formatul numeric se scrie in maniera Basic!!!
        .Plot.Axis(1).Labels.Item(i).Format = "###,###,##0"
    ENDFOR
    * pozitia legendei (vezi directiva #INCLUDE si fisierul Pozitii_legenda.txt)
    .Legend.Location.LocationType=LegendaPozJosCentru
```



```
    * stabilirea numelor seriilor de date
    .Column=1 && pozitionare pe coloana 1
    .ColumnLabel='Valoare fara TVA' && eticheta pentru coloana 1
    .Column=2 && pozitionare pe coloana 2
    .ColumnLabel='Valoare cu TVA' && eticheta pentru coloana 2
    * titlul graficului
    .TitleText='Situatia vinzarii produselor pe zile'
ENDWITH
```

După cum se observă, formatul cifrelor pe axa `Oy` implică un anume grad de dificultate:

- axele de coordonate aparțin unui alt obiect inclus în grafic, numit `Plot` (este de fapt o proprietate-obiect a graficului);
- fiecare axă este identificată printr-un indice: `Axis(1)` este axa `Oy`;
- fiecare diviziune de pe axă are asociat un obiect de tip `Label`, identificat printr-un indice: `Labels.Item(i)`, unde `i` variază de la `1` la `Labels.Count`; așadar, pentru a modifica toate etichetele, le parcurgem într-o buclă `FOR...ENDFOR`;
- modificarea aspectului etichetei se face prin atribuirea unui șablon proprietății `Format` a acesteia: `Labels.Item(i).Format="###,###,##0"`[46].

O atenție deosebită merită și instrucțiunea

```
.Legend.Location.LocationType=LegendaPozJosCentru
```

întrucât pentru obiectul `Legend` (la rândul său inclus în obiectul `Chart`), proprietatea `Location` poate lua valori de la 0 (poziția stânga-sus) până la 8 (poziție definită de utilizator). Pentru a ușura folosirea acestor valori în Visual FoxPro, le putem declara constante în fișierul `pozitii_legenda.txt` (vezi figura 10.17).

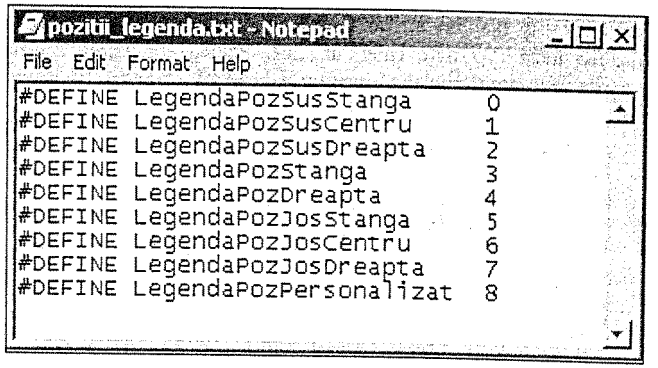


Figura 10.17. Constante pentru poziția legendei unui obiect `Chart`

Acest fișier se adaugă formularului prin directiva[47]

```
#INCLUDE "\aplicatie\pozitii_legenda.txt"
```

plasată la începutul evenimentului `Init`. Astfel, este posibil ca în loc de cifra 6 să folosim numele constantei corespunzătoare: `LegendaPozJosCentru`.

46. Șablonul trebuie scris în manieră Basic (sau Excel): un diez ține locul unei cifre, virgula separă grupele de trei cifre, punctul semnifică marca zecimală, 0 înseamnă că, dacă numărul este 0 sau subunitar, prima cifră afișată este zero.
47. Vezi și capitolul 4, paragraful 4.3.

După completările aduse formularului, graficul afişat ar trebui să arate ca în figura 10.18.

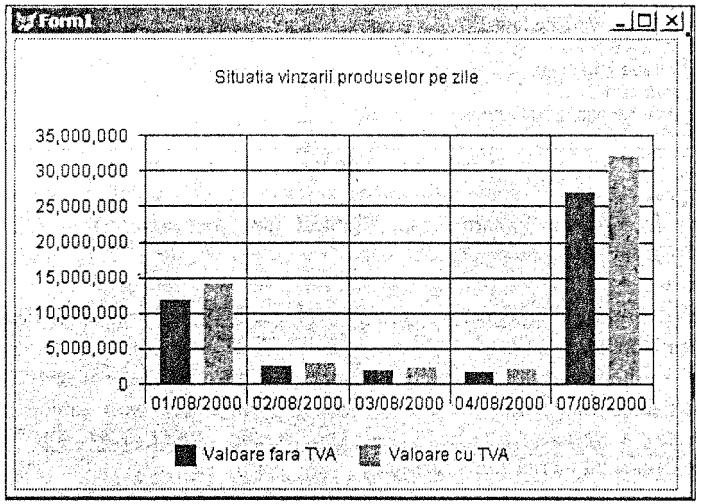


Figura 10.18. Varianta rafinată a graficului vânzărilor pe zile

10.5.3. Controlul `Calendar` sau o altă modalitate de lucru cu datele calendaristice

Controlul `Calendar` este un alt control `ActiveX` pe care-l putem utiliza într-un formular ce implică lucrul cu date calendaristice. Pentru adăugarea acestui control pe formular, vom urma recomandările din paragraful anterior, doar că, de această dată, vom selecta `Calendar Control 8.0`. Se va obţine imaginea din figura 10.19.

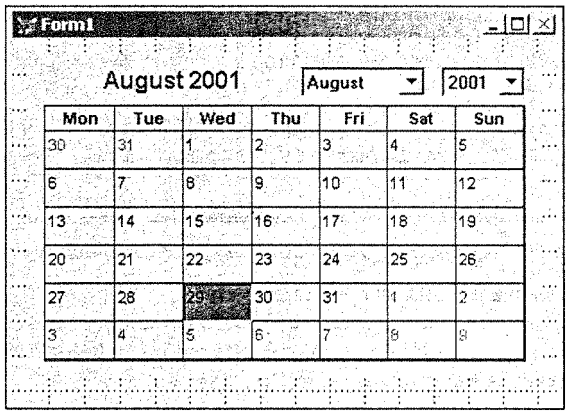


Figura 10.19. Controlul `Calendar`

Controlul prezintă următoarele particularităţi:

- după poziţionarea pe anul şi luna dorite, calendarul lunii în curs se afişează în porţiunea tabelară a controlului;
- putem selecta o zi cu ajutorul mouse-ului sau al tastelor săgeţi;
- în cazul utilizării tastaturii, după ultima zi din lună va fi afişată automat luna următoare, iar înainte de prima zi din luna curentă se trece automat la luna precedentă.

Tabelul 10.4. Principalele proprietăţi şi evenimente ale controlului `Calendar`

Proprietate	*Semnificaţie*
`Day`	Ziua (disponibilă pentru citire/scriere)
`Month`	Luna (disponibilă pentru citire/scriere)
`Year`	Anul (disponibilă pentru citire/scriere)
`AfterUpdate`	Survine după schimbarea datei curente
`BeforeUpdate`	Survine înainte de schimbarea datei curente
`PreviousDay`	Determină deplasarea cu o zi înapoi
`NextDay`	Determină deplasarea cu o zi înainte
`PreviousMonth`	Determină deplasarea cu o lună înapoi
`NextMonth`	Determină deplasarea cu o lună înainte
`PreviousYear`	Determină deplasarea cu un an înapoi
`NextYear`	Determină deplasarea cu un an înainte

Data selectată la un moment dat se poate afla cu ajutorul funcţiei `DATE(<nume-control>.Year, <nume-control>.Month, <nume-control>.Day)`. De remarcat că manipularea controlului `Calendar` se poate efectua atât prin acţiuni asupra mouse-ului şi tastaturii, cât şi prin cod, ţinând seama de proprietăţile şi metodele prezentate în tabelul 10.4. De exemplu, pentru a „selecta" data de 20 mai 1990 sunt necesare următoarele instrucţiuni de atribuire:

```
<nume-control>.Day=20
<nume-control>.Month=5
<nume-control>.Year=1990
```

Controlul `Calendar` se utilizează, în general, pentru filtrarea datelor în rapoarte. Pornind de la exemplul din subcapitolul anterior, dorim să oferim utilizatorului posibilitatea de a-şi stabili singur perioada de timp pentru care vor fi reprezentate grafic vânzările. În acest scop, formularul `frmCalendar` (figura 10.20) va cuprinde următoarele elemente:

- controlul de tip `Calendar`, numit `oleCalendar`;
- un control de tip `Label` pentru data iniţială (`lblDataStart`) şi încă unul pentru data finală (`lblDataStop`);
- un buton de comandă numit `cmdReset`, cu rolul de a „şterge" cele două date din etichetele `lblDataStart` şi `lblDataStop`;
- un buton numit `cmdProceed`, care va lansa în execuţie formularul `frmChart`.

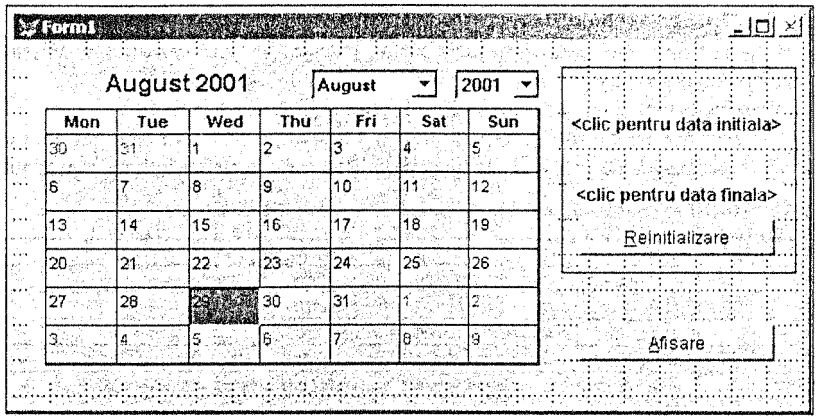


Figura 10.20. Aspectul formularului `frmCalendar`

Metodele controalelor formularului `frmCalendar` sunt redate în listingul 10.9. Se observă că cele două controale `lblDataStart` şi `lblDataStop` vor constitui parametrii formularului `frmChart`.

Listing 10.9. Codul-sursă al metodelor formularului `frmCalendar`

```
                                *Obiect: lblDataStart
*Eveniment: Click
WITH THISFORM.olecalendar
* se atribuie ca zi implicita de pornire ziua de 1 ale lunii
    .DAY=1
    THIS.CAPTION="<alegeti din calendar>"
ENDWITH

                                *Obiect: lblDataStop
*Eveniment: Click
WITH THISFORM.olecalendar
* se atribuie ca zi implicita de oprire ziua curenta
    .DAY=DAY(DATE())
    THIS.CAPTION="<alegeti din calendar>"
ENDWITH

                                *Obiect: oleCalendar
*Eveniment: BeforeUpdate
*** ActiveX Control Event ***
LPARAMETERS CANCEL
IF THISFORM.lbldatastop.VISIBLE=.T. && inseamna ca se alege data de stop
                                    && vezi evenimentul lblDataStart.Click
    THISFORM.lbldatastop.CAPTION= ;
        DTOC(DATE(THIS.YEAR,THIS.MONTH,THIS.DAY))
* se dezactiveaza calendarul, pentru a nu mai putea schimba vreo data
    THIS.ENABLED=.F.
ELSE                                && inseamna ca se alege data de start
    THISFORM.lbldatastart.CAPTION= ;
        DTOC(DATE(THIS.YEAR,THIS.MONTH,THIS.DAY))
* se afiseaza eticheta pentru data de stop, pentru a o putea alege din calendar
    THISFORM.lbldatastop.VISIBLE=.T.
ENDIF
```



```
                                *Obiect: cmdReset
*Eveniment: Click
THISFORM.lblDataStart.CAPTION="<clic pentru data initiala>"
THISFORM.lblDataStop.CAPTION="<clic pentru data finala>"
THISFORM.lblDataStop.VISIBLE=.F.
THISFORM.olecalendar.ENABLED=.T.

                                *Obiect: cmdProceed
*Eveniment: Click
THISFORM.RELEASE
WAIT WIND "Va rog asteptati..." NOWAIT
DO FORM "\aplicatie\forms\frmchart.scx" ;
    WITH CTOD(THISFORM.lbldatastart.CAPTION), ;
    CTOD(THISFORM.lbldatastop.CAPTION)
```

La rândul său, formularul `frmChart` se parametrizează[48], pentru a putea selecta doar vânzările aferente perioadei dintre două date calendaristice transmise de `frmCalendar` (vezi listingul 10.10). În acelaşi eveniment `Init` va trebui să testăm dacă cei doi parametri sunt într-adevăr de tip dată calendaristică (este posibil ca formularul `frmChart` să fie lansat şi independent, fără parametri, caz în care va prelucra *toate* facturile).

Listing 10.10. Versiunea parametrizată a procedurii-eveniment `frmChart.Init`

```
LPARAMETERS datastart_, datastop_
#INCLUDE "\aplicatie\pozitii_legenda.txt"

IF !DBUSED('vinzari')
    OPEN DATABASE '\aplicatie\database\vinzari' SHARED
ENDIF

* daca cei doi parametri contin date calendaristice valide, mai efectuam o filtrare
IF VARTYPE (datastart_)="D" AND VARTYPE (datastop_)="D"
    SELECT datafact, SUM(cantitate*pretunit) AS Val_zi_fTVA, ;
        SUM(cantitate*pretunit*1.19) AS Val_zi_cuTVA ;
        FROM facturi F inner JOIN liniifact lf ON F.nrfact=lf.nrfact ;
        and datafact BETWEEN datastart_ AND datastop_ ;
        INTO CURSOR sursa_grafic ;
        GROUP BY datafact ;
        ORDER BY datafact
* titlul graficului este si el personalizat:
    THISFORM.olechart.TitleText='Situatia vinzarii produselor intre ' ;
        + DTOC(datastart_)+" si " +DTOC(datastop_)
ELSE
    SELECT datafact, SUM(cantitate*pretunit) AS Val_zi_fTVA, ;
        SUM(cantitate*pretunit*1.19) AS Val_zi_cuTVA ;
        FROM facturi F inner JOIN liniifact lf ON F.nrfact=lf.nrfact ;
        INTO CURSOR sursa_grafic ;
        GROUP BY datafact ;
        ORDER BY datafact
* titlul graficului este simplu
    THISFORM.olechart.TitleText='Situatia vinzarii produselor pe zile'
ENDIF

SELECT sursa_grafic
```

48. Vezi paragraful 10.4 al acestui capitol.

```
THISFORM.olechart.COLUMNCOUNT=2 && graficul sa aiba doua serii de date:
&& valoarea fara TVA si valoarea cu TVA
* graficul are atatea categorii cate linii are tabela-sursa:
THISFORM.olechart.Rowcount=RECCOUNT('sursa_grafic')
* popularea cu date folosind o bucla SCAN pe tabela SURSA_GRAFIC
SCAN
    WITH THISFORM.olechart
        .ROW=RECNO() && pozitionare pe categoria care corespunde liniei curente din tabela
        .COLUMN=1 && pozitionare pe coloana (seria de date) 1, linia curenta
        .DATA=sursa_grafic.Val_zi_fTVA && atribuire sursa de date pentru coloana 1
        .COLUMN=2 && pozitionare pe coloana (seria de date) 2, linia curenta
        .DATA=sursa_grafic.Val_zi_cuTVA && atribuire sursa de date pentru coloana 2
        .RowLabel=ALLTRIM(DTOC(datafact))
    ENDWITH
ENDSCAN
* atribute vizuale ale graficului
WITH THISFORM.olechart
* se stabileste formatul etichetelor de pe axa Oy, pentru afisarea milioanelor
    FOR i=1 TO .Plot.Axis(1).Labels.COUNT
* se scrie in maniera Basic!!!
        .Plot.Axis(1).Labels.ITEM(i).FORMAT = "###,###,##0"
    ENDFOR
* pozitia legendei
    .Legend.Location.LocationType=LegendaPozJosCentru
    .COLUMN=1 && pozitionare pe coloana 1
    .ColumnLabel='Valoare fara TVA' && eticheta pentru coloana 1
    .COLUMN=2 && pozitionare pe coloana 2
    .ColumnLabel='Valoare cu TVA' && eticheta pentru coloana 2
ENDWITH
```

Execuția celor două formulare „înlănțuite" este ilustrată în figura 10.21.

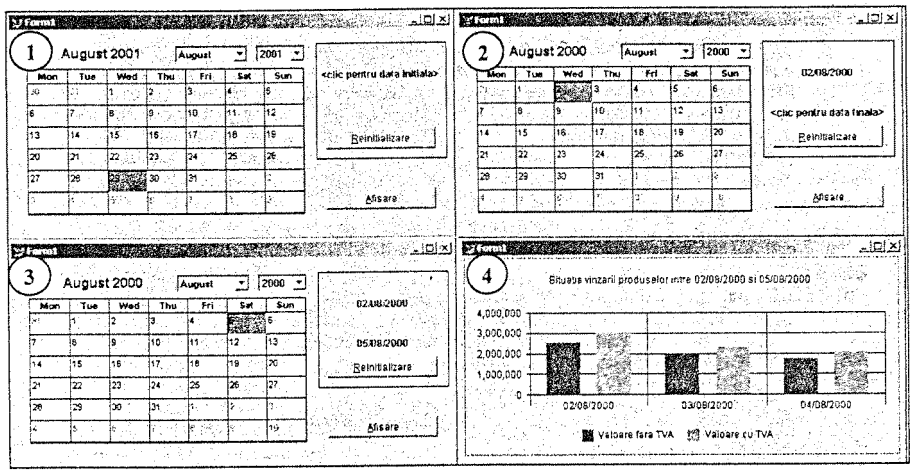


Figura 10.21. Pașii selecției datelor calendaristice și realizării graficului parametrizat

10.5.4. Controlul `ProgressBar`

`ProgressBar` este un control `ActiveX` ce poate fi utilizat pentru a indica starea execuției unei operații de durată. Astfel de indicatoare întâlnim în majoritatea aplicațiilor Windows atunci când au loc acțiuni precum copierea sau descărcarea de pe Internet a unor fișiere, instalarea unei noi aplicații etc. Proiectantul unei aplicații cu baze de date poate utiliza respectivul indicator pentru a afișa gradul de îndeplinire a unor operații de populare ori chiar parcurgere a unor tabele cu un volum mare de date, efectuarea unor calcule laborioase etc. În formulare, putem combina acest control cu forma de clepsidră a cursorului mouse-ului (`thisform.mousepointer=13` sau `_screen.mouse pointer=13`). Pe lângă aspectul pur estetic, utilizatorul final sesizează mai ușor dacă aplicația rulează o operațiune mai complexă sau pur și simplu s-a blocat...

Practic, aspectul vizual al acestui control este acela al unui dreptunghi care se umple, de la stânga la dreapta, cu o culoare, pe măsură ce procesul monitorizat se apropie de sfârșit (figura 10.22).

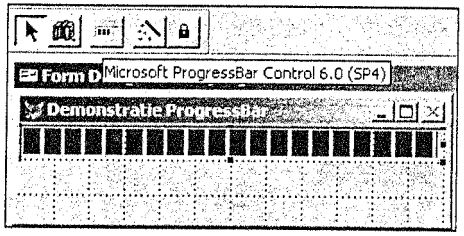


Figura 10.22. Controlul `ProgressBar`

Proprietate	*Semnificație*
`Align`	Permite alinierea automată la una din laturile formularului-gazdă.
`Max`	Valoarea maximă admisibilă (capătul din dreapta).
`Min`	Valoarea minimă admisibilă (capătul din stânga).
`Orientation`	0 – orizontală, 1 – verticală (cu efectul de umplere de jos în sus).
`Scrolling`	Aspectul porțiunii colorate: în „felii" (0) sau continuu (1).
`Value`	Valoarea curentă. Se poate stabili numai prin cod.

Pe formularul numit `frmProgress` adăugăm un control de tip `Label` numit `lblProcent`, în care vom afișa (în cifre) procentul de realizare a operației. Plasăm și un buton cu numele `cmdStart`, al cărui eveniment `Click` este destinat să lanseze o procedură de test. Procedura (listingul 10.11) constă în crearea unei tabele temporare și popularea ei cu un număr de 2.500.000 de înregistrări (șiruri de caractere aleatorii). La anumite intervale de timp se consultă raportul procentual dintre numărul de înregistrări adăugate și cel ce se intenționează a se adăuga. Rezultatul acestui raport este afișat de controlul `lblProcent` și totodată servește la actualizarea proprietății `Value` a

controlului `oleProgress`. De menționat că, în loc să se modifice proprietățile `Min` și `Max`, s-a preferat transformarea numărului de reprezentat într-un procent. Ca element estetic, formularul are proprietatea `AutoCenter=.T.`

Listing 10.11. Secvență de cod demonstrativă pentru folosirea controlului `ProgressBar`

```
* se creeaza o tabela temporara care va fi populata cu
* un numar de 2500000 de inregistrari, completate arbitrar
CREATE CURSOR test(f1 c(10), f2 c(12))
nrmax=2500000
FOR curenta=1 TO nrmax
* functia sys(2015) creeaza un sir de caractere unic
    INSERT INTO test VALUES (SYS(2015), "A"+STR(curenta))
* numarul de inregistrari adaugate se converteste in procent extragandu-se doar partea intreaga
    procent=INT(curenta*100/nrmax)
* indicatorul va fi actualizat numai la 1%, 2%... (se exclud procentele cu zecimale)
* altfel timpul de executie este mult mai mare
    IF procent=curenta*100/nrmax
        THISFORM.oleProgress.VALUE=procent
        THISFORM.lblProcent.CAPTION=ALLTRIM(STR(procent))+"%"
    ENDIF
ENDFOR
MESSAGEBOX("Gata!")
THISFORM.RELEASE
```

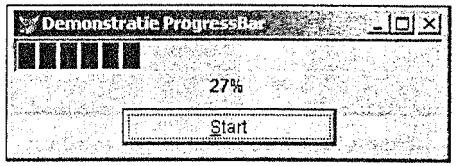


Figura 10.23. Rezultatul execuției formularului `frmProgress`

În practică vom găsi suficiente ocazii pentru a folosi acest control, dar de multe ori modulul care execută actualizări mari consumatoare de timp nu face parte dintr-un formular, ci dintr-un program extern. Cum se accesează atunci un eventual control `ProgressBar` ? Ei bine, primul pas este crearea unui mic formular care să cuprindă doar acest control, eventual fără bară de titlu și butoane de control (`TitleBar=0`) și cu bordură fixă (`BorderStyle=2`). Presupunem că acest formular se numește tot `frmProgress` și are (la execuție) aspectul din figura 10.24. Secvența de cod „consumatoare de timp" este prezentată în listingul 10.12, fiind de data aceasta memorată într-un program oarecare.

Listing 10.12. Exemplu de utilizare a controlului `ProgressBar` – varianta 2

```
* se lanseaza formularul frmProgress
DO FORM "\aplicatie\forms\frmProgress"
* se creeaza o tabela temporara care va fi populata cu
* un numar de 2500000 de inregistrari, completate arbitrar
CREATE CURSOR test(f1 c(10), f2 c(12))
nrmax=2500000
FOR curenta=1 TO nrmax
* functia sys(2015) creeaza un sir de caractere unic
    INSERT INTO test VALUES (SYS(2015), "A"+STR(curenta))
* numarul de inregistrari adaugate se converteste in procent
    procent=INT(curenta*100/nrmax)
```



```
* indicatorul va fi actualizat numai la 1%, 2%... (se exclud procentele cu zecimale)
* altfel timpul de executie este mult mai mare
    IF procent=curenta*100/nrmax
        _SCREEN.ACTIVEFORM.oleProgress.VALUE=procent
        _SCREEN.ACTIVEFORM.lblProcent.CAPTION=ALLTRIM(STR(procent))+"%"
    ENDIF
ENDFOR
MESSAGEBOX("Gata!")
_SCREEN.ACTIVEFORM.RELEASE
```

De remarcat modalitatea de acces la obiectele unui formular aflat în execuție prin notația `_Screen.ActiveForm` (vezi și subcapitolul 10.4).

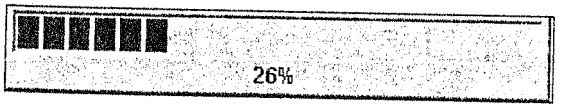


Figura 10.24. Versiune simplificată a formularului `frmProgressBar`

Atenție! În manipularea programatică a controlului `ProgressBar` trebuie avut grijă ca niciodată să nu se atribuie proprietății `Value` o valoare în afara intervalului `Min...Max`. În caz contrar, survine o eroare „fatală".

10.5.5. Controalele `ImageList`, `TreeView` și `ListView`

Cele trei controale pe care le vom aborda în cele ce urmează fac parte din așa-zisa categorie *Windows Common Controls*. Termenul „comun" este cât se poate de adecvat, dacă ne gândim că aceste controale se numără printre „cărămizile" de bază ale interfeței grafice Windows. Dacă ar fi să explicăm aceste controale prin exemple edificatoare, am recurge cu siguranță la bine cunoscuta fereastră a gestionarului de fișiere Windows Explorer. Acesta prezintă în partea stângă partițiile discurilor și directoarele într-o structură arborescentă (ierarhică), cu „rădăcini", „ramuri" și „frunze", și are drept corespondent, în cazul de față, controlul `TreeView`. Fereastra din dreapta a aceluiași Windows Explorer afișează (în diverse moduri) subdirectoarele și fișierele elementului selectat în fereastra din stânga și corespunde controlului `ListView`. Tot controlul `ListView` este prezent și în ferestrele de dialog pentru deschiderea fișierelor, utilizate de o multitudine de aplicații Windows, printre care și Visual FoxPro. Cât despre controlul `ImageList`, acesta servește drept vector în care se păstrează referințe la diverse fișiere grafice, care pot fi folosite ca pictograme pentru butoanele unei bare de instrumente, ca element distinctiv pentru ramurile controlului `TreeView` ori ca pictograme pentru elementele controlului `ListView`. Funcționalitatea controalelor `TreeView` și `ListView` este cât se poate de utilă și totodată spectaculoasă.

Să presupunem că, imitând maniera de afișare Windows Explorer, dorim să proiectăm un formular care să arate ca în figura 10.25.

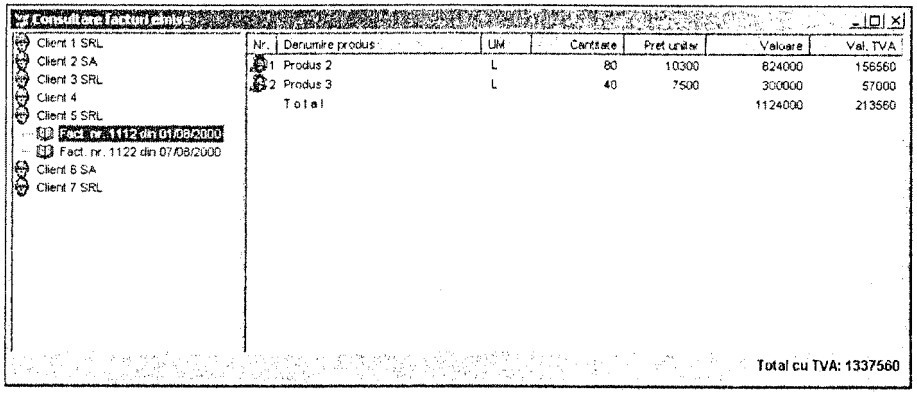


Figura 10.25. Afişarea la cerere a detaliilor unei facturi

Deşi realizarea unui astfel de formular nu presupune o abordare tehnică foarte complicată, ar fi totuşi indicat a se consulta documentaţia controalelor respective, precizările pe care le vom oferi aici rezumându-se la strictul necesar realizării formularului din figura 10.26.

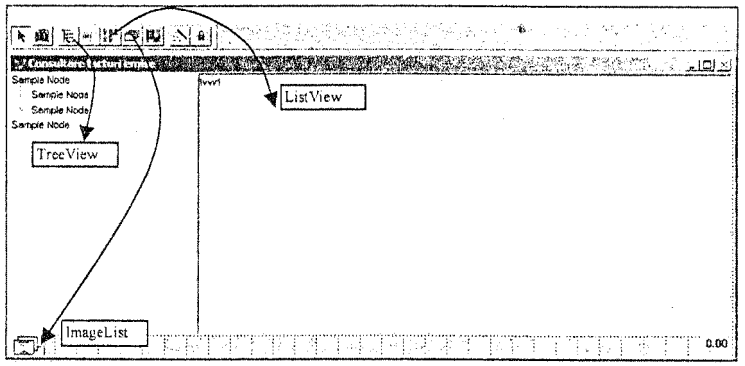


Figura 10.26. Formularul `frmDetaliiFacturi` în faza de proiectare

În figura 10.26 prezentăm formularul (numit `frmDetaliiFacturi`) în etapa de proiectare[49]. Se observă pe suprafaţa lui cele trei controale, de tip `ImageList`, `TreeView` şi `ListView`, inserate în prealabil pe bara de instrumente `ActiveX` cu ajutorul opţiunii de meniu `Tools→Options→Controls`. După dimensionarea lor corespunzătoare, pe formular se mai adaugă un control de tip `Label` (dreapta-jos), pentru afişarea valorii totale a facturii selectate (vezi şi figura 10.25).

Pentru controlul `ImageList`, la fel ca şi pentru `Timer`, poziţia şi dimensiunile definite la creare nu au importanţă, întrucât controlul de tip `ImageList` nu apare la execuţie. Imaginile pe care le gestionează acesta pot fi afişate în alte controale.

49. Vezi şi paragrafele anterioare, pentru modalitatea de inserare a unui control `ActiveX` pe un formular.

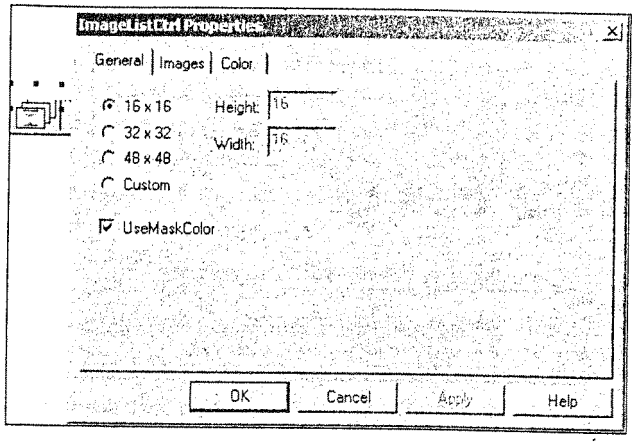


Figura 10.27. Controlul `ImageList` şi câteva dintre proprietăţile sale

Dimensiunile şi conţinutul listei de imagini se pot defini prin intermediul proprietăţilor obiectului (din meniul contextual, opţiunea `ImageListCtrl Properties`), aşa cum putem observa în figura 10.27. În pagina `Images` specificăm lista imaginilor de afişat. Se utilizează butonul `Insert Picture`, care deschide o casetă de dialog pentru selectarea imaginii dorite[50]. Controlul `ImageList`, numit `Iml1`, recunoaşte fişiere `.ico`, `.bmp`, `.jpg` şi `.gif`. Se preferă fişierele `.ico`, deoarece:

- afişarea lor consumă puţină memorie;
- au un număr redus de culori, deci se vor afişa inteligibil pe ecranul oricărui sistem de calcul;
- au dimensiunile standard de 16×16 pixeli (*Small Icons*), 32×32 (*Large Icons*) ori 48×48, ceea ce duce la încadrarea optimă pe butoanele de comandă;
- pot avea porţiuni transparente, spre deosebire de alte formate grafice, pentru care fundalul este un dreptunghi alb.

Observaţie. După popularea controlului `ImageList` nu se mai pot modifica dimensiunile pictogramelor din fereastra `General`.

50. O bună sursă pentru astfel de fişiere este directorul `C:\Program Files\Microsoft Visual Studio\Graphics`. Numărul de fişiere existente depinde de opţiunile selectate la instalarea pachetului Visual Studio.

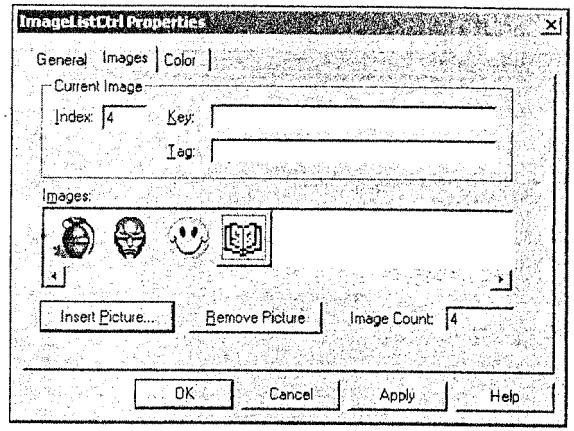


Figura 10.28. Imaginile folosite la popularea controlului `iml1`

Controlul `TreeView` (cel de pe formularul în lucru este numit `tvw1`) este un tip special de listă ale cărei elemente, numite „noduri" (*nodes*), pot fi ierarhizate, utilizând opțiuni specifice, în etapa de populare. Nodurile-copil sunt implicit comprimate (invizibile) și pot fi expandate prin dublu-click pe un nod (dar nu și eventualii „nepoți"). Nodurile de nivel mai mic vor fi afișate prin deplasare spre dreapta cu câțiva pixeli (vezi proprietatea `Indent`). În faza de proiectare a formularului, controlul de tip `TreeView` nu afișează date, ci doar o schiță a „arborelui". Folosind proprietatea `Style`, se poate decide cum vor fi afișate nodurile. Sunt șapte valori posibile pentru această proprietate. În plus, proprietatea `Checkboxes` permite (dacă are valoarea `.T.`) ca fiecare nod să fie însoțit de o casetă de opțiune, utilă în operații de selecție.

Popularea arborelui este posibilă numai în momentul execuției, prin invocarea metodei `Add`. Aceasta aparține colecției `Nodes` (o proprietate a acestui tip de control) și poate fi folosită utilizând următoarea sintaxă:

```
<nume-obiect>.Nodes.Add ([<cheie-nod-părinte>], [<relație-ierarhică>], <cheie-nod-curent>, <text-nod-curent>, <indice-imagine>, <indice-imagine-selectată>)
```

Semnificațiile argumentelor sunt următoarele:

- `cheie-nod-curent`: un șir de caractere folosit ca identificator unic pentru nodul curent;
- `text-nod-curent`: un șir de caractere afișat pe nodul curent;
- `cheie-nod-părinte`: cheia nodului față de care se dorește ierarhizarea nodului curent;
- `relație-ierarhică`: un număr cu următoarele semnificații: 0 – primul nod, 1 – ultimul nod, 2 (implicit) – după nodul anterior, 3 – deasupra nodului anterior, 4 – copil al nodului anterior;
- `indice-imagine` – indicele imaginii dintr-un control de tip `ImageList`, populat în prealabil ca în figura 10.28, ce se dorește a fi afișată de nodul respectiv;
- `indice-imagine-selectată` – indicele imaginii dintr-un control de tip `ImageList`, ce se dorește a fi afișată de nod atunci când acesta a fost selectat prin click.

Pentru a putea folosi ultimele două argumente ale metodei `Add`, este necesar ca, înainte de popularea controlului `TreeView`, să se stabilească valorile proprietății `ImageList` astfel încât aceasta să se refere la un control corespunzător care există și conține imagini. De asemenea, proprietatea `Style` trebuie să aibă una din valorile 1, 3, 5 sau 7.

Fiecare nod are un indice corespunzător ordinii sale de adăugare. Referirea programatică la un nod anume se poate face prin indice (`Nodes(indice)`), prin cheie (`Nodes(cheie)`) ori prin relațiile sale „de rudenie" (vezi tabelul 10.5).

Tabelul 10.5. Proprietăți ale obiectelor de tip `Node`

Proprietate	*Semnificație*
`Parent`	Referință la nodul-părinte (dacă nu are, atunci `VARTYPE(<nod>.Parent)<>"O"`).
`Children`	Read-only. Returnează numărul de noduri-copil ale unui nod dat.
`Child(<indice>)`	Permite adresarea unui nod-copil de la nivelul unui nod-părinte. Indicele depinde de ordinea adăugării. Indicele ultimului copil este egal cu valoarea proprietății `Children`.
`Expanded`	Permite expandarea (`.T.`) sau comprimarea (`.F.`) programatică a ierarhiei de noduri de sub un nod dat sau aflarea situației de expandare/comprimare.
`Checked`	Permite „bifarea" (`.T.`) sau „debifarea" (`.F.`) programatică a unui nod dat, sau aflarea situației sale de „bifare". Funcționează numai dacă proprietățile `Style` și `Checkboxes` ale controlului permit și afișarea de căsuțe de opțiune în stânga textului nodului.

Ștergerea unui nod căruia i se cunoaște indicele are loc prin apelul metodei `<nume-obiect>.Nodes.Remove(indice)`. Ștergerea tuturor nodurilor presupune invocarea metodei `<nume-obiect>.Nodes.Clear`.

Controlul de tip `TreeView` este deosebit de bogat în evenimente, dar în exemplul de față nu am folosit decât evenimentul `NodeClick`. Cât despre popularea controlului, aceasta se face astfel:

- într-o tabelă temporară (cursor) se obțin clienții cărora li s-au emis facturi, împreună cu numerele și datele facturilor;
- tabela se parcurge la inițializarea formularului, fiecare client distinct fiind adăugat ca nod de nivel 1, pe când facturile sunt adăugate ca noduri-copil pentru clientul către care au fost emise.

Controlul `ListView` este o altă variantă de listă, ale cărei elemente pot fi prezentate în mai multe moduri (pe care le cunoaștem din utilitarul Windows Explorer), după cum se arată în figura 10.29. Diferitele modalități de afișare depind de valoarea proprietății `View`, ce poate fi `0 - Icon, 1 - SmallIcon, 2 - List, 3 - Report`.

Implicit, o asemenea listă are o singură coloană, dar colecția `ColumnHeaders` poate fi manevrată în scopul adăugării de noi coloane, atât din fereastra specifică (se activează din meniul contextual, prin opțiunea `ListViewCtrl Properties` – vezi figura 10.30), cât și prin program.

Dacă în cazul controlului `TreeView` elementele afișate erau referite prin colecția `Nodes`, la `ListView` dispunem de colecția `ListItems`, cu metodele corespunzătoare `Add`, `Remove` și `Clear`. În cazul stilului de afișare `Report`, fiecare element dispune de colecția `SubItems`.

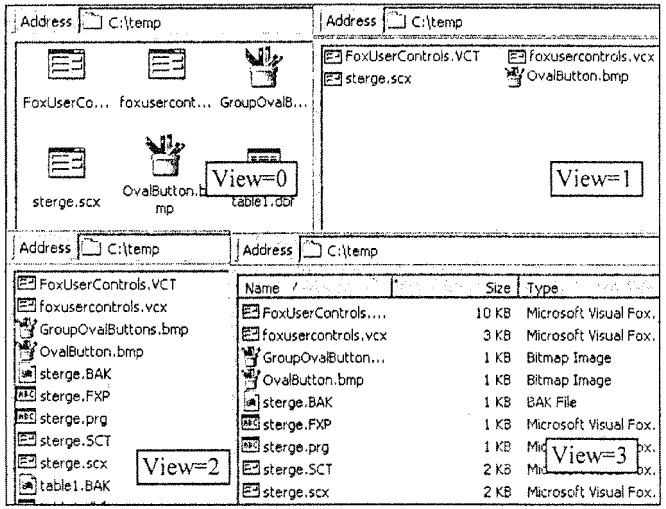


Figura 10.29. Patru moduri de afişare corespunzătoare proprietăţii `View` a controlului `ListView`

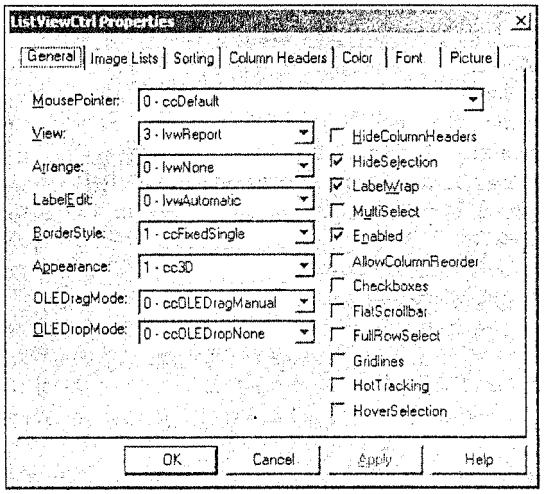


Figura 10.30. Câteva proprietăţi ale controlului de tip `ListView`

Proprietăţile şi metodele controlului de tip `ListView` sunt foarte numeroase şi pot fi aprofundate la nevoie consultând documentaţia aferentă (selectaţi un astfel de control din formular şi tastaţi F1[51]).

Controlul de tip `ListView` din formularul nostru a fost denumit `lvw1` şi este destinat afişării detaliilor unei facturi selectate în prealabil în controlul de tip `TreeView`, `tvw1`, în

51. Pentru controalele de tip `ActiveX` este necesară instalarea completă a documentaţiei Visual Studio.

manieră asemănătoare cu a controlului intrinsec `Grid`, dar fără posibilitatea modificării datelor. În faza de execuţie, lucrurile vor decurge în modul următor:

- la iniţializarea formularului, controlului `lvw1` (cu stilul de afişare `3 - Report`) i se adaugă un număr de coloane suficient pentru a afişa: numărul liniei din factură, denumirea produsului, unitatea de măsură, cantitatea, preţul unitar, valoarea (cantitatea × preţul unitar) şi valoarea TVA colectată;
- la selecţia unui număr de factură din controlul `tvw1` (se foloseşte evenimentul `NodeClick`) se obţin într-un cursor detaliile facturii alese, apoi lista `lvw1` se populează prin parcurgerea cursorului.

Adăugarea de coloane la un control de tip `ListView` are loc cu ajutorul metodei `Add` a colecţiei `ColumnHeaders`, metodă ce are următorul format:

```
<listă>.ColumnHeaders.Add([<indice>], [<cheie-coloana>], <text-antet>, <lăţime>, [<aliniere>])
```

Semnificaţiile argumentelor sunt următoarele:

- `indice` : număr ce arată poziţia coloanei;
- `cheie-coloană`: un şir de caractere folosit ca identificator unic pentru coloană (indicele se schimbă dacă se mută coloanele în timpul execuţiei, cu ajutorul mouse-ului);
- `text-antet`: textul afişat în antetul coloanei;
- `lăţime` – dimensiunea pe orizontală a coloanei, în pixeli (o soluţie bună de dimensionare proporţională este instituirea unei unităţi convenţionale, prin raportarea la proprietatea `Width` a controlului de tip `ListView`);
- `aliniere`: stilul alinierii elementelor care vor popula coloana: `0 - Left` (implicit), `1 - Right`, `2 - Center`. Pentru prima coloană *nu* se poate stabili decât aliniere la stânga.

Adăugarea de elemente (linii) într-un control de tip `ListView` are loc prin invocarea următoarelor metode:

```
<nume-obiect>.ListItems.Add ([<indice>], <cheie-element-curent>, <text-element-curent>, <indice-imagine>, <indice-imagine-mică>)
```

Semnificaţiile argumentelor sunt următoarele:

- `indice` : număr ce arată poziţia elementului în listă;
- `cheie-element-curent`: un şir de caractere folosit ca identificator unic pentru elementul curent;
- `text-element-curent`: un şir de caractere afişat ca element curent;
- `indice-imagine` – indicele imaginii dintr-un control de tip `ImageList`, populat în prealabil ca în figura 10.28, ce se doreşte a fi afişată în stânga textului elementului, atunci când stilul de afişare este `LargeIcons` (`View=0`); necesită ca proprietatea `Icons` a controlului de tip `ListView` să indice un control `ImageList` existent şi populat;
- `indice-imagine-mică` – indicele imaginii dintr-un control de tip `ImageList`, ce se doreşte a fi afişată în stânga textului elementului atunci când stilul este `SmallIcons` (`View=1`), `List` (`View=2`) sau `Details` (`View=3`); necesită ca proprietatea `SmallIcons` a controlului de tip `ListView` să indice un control `ImageList` existent şi populat cu pictograme de 16×16 pixeli.

Această metodă produce adăugarea de text numai în prima coloană (cea implicită) a controlului de tip `ListView` (în cazul nostru, ea ar fi suficientă pentru adăugarea la listă a numerelor de linie din factură). Tot în această coloană se va afişa şi pictograma precizată prin argumentul `<indice-imagine>` ori, după caz, `<indice-imagine-mică>`. Pentru a adăuga alte şiruri de caractere sau numere (în cazul de faţă, denumirea produsului, cantitatea etc.) pe coloanele următoare, este nevoie de parcurgerea ciclică a colecţiei `SubItems`. Sunt atâtea „sub-elemente" câte coloane are lista, mai puţin una. Accesarea unui asemenea element de nivel inferior se face în maniera:

`<element-din-listă>.SubItems(indice)=<valoare>`

Cel din a doua coloană are indicele 1.

O imagine mai clară asupra celor prezentate se va contura, sperăm, după parcurgerea codului formularului `frmDetaliiFacturi`, reprodus în listingul următor.

Listing 10.13. Evenimentele şi metodele formularului `frmDetaliiFacturi`

```
*Obiect: frmDetaliiFacturi
* Eveniment: Init
IF !DBUSED('vinzari')
    OPEN DATABASE "\aplicatie\database\vinzari" SHARED
ENDIF
SET EXCLUSIVE OFF
* mai intai, se populeaza controlul de tip TreeView
* cu toti clientii (nivel 1)  care au facturi si cu facturile in ordinea cronologica
SELECT cl.codcl, dencl, nrfact, datafact ;
    FROM clienti cl, facturi F ;
    INTO CURSOR cFactEmise ;
    WHERE F.codcl=cl.codcl ;
    ORDER BY 2,4
IF _TALLY<>0
    SELECT cFactEmise
    * se atribuie dinamic proprietatea ImageList
    THISFORM.tvw1.imagelist=THISFORM.iml1
    SCAN
        WITH THISFORM.tvw1
            * se adauga o ramura de nivel 1
            IF .NODES.COUNT=0 && initial nu sunt noduri
                * nod-parinte, cu 2 pictograme pentru starile "normal" si "selectat"
                *  care sunt imaginile 2 si 3 din controlul de la proprietatea
tvw1.imagelist
                * virgulele si spatiile arata argumente optionale omise
                * s-a concatenat o tilda (~) pentru ca valoarea cheii
                * sa nu inceapa cu cifra, intrucat e incorect
                .NODES.ADD (, ,"~"+ALLT(STR(codcl)), ALLT(dencl),2,3)
                * nod-copil, cu pictograma nr 4 de la proprietatea tvw1.imagelist
                .NODES.ADD ("~"+ALLT(STR(codcl)),4,"~"+ALLT(STR(nrfact)) , ;
                        "Fact. nr. " + ALLT(STR(nrfact)) + " din " +DTOC(datafact),4)
                * se memoreaza clientul pentru a nu-i mai adauga noduri cu acelasi
cod
                * intrucat este incorect (cheia e unica!)
                codclanterior=codcl
            ELSE && s-au adaugat noduri
                IF codclanterior<>codcl && s-a schimbat clientul?
                    * un nou nod-parinte (cu tot cu pictograme)
                    .NODES.ADD (, ,"~"+ALLT(STR(codcl)), ALLT(dencl),2,3)
                    codclanterior=codcl
                ENDIF
                * nod-copil, cu pictograma nr 4 de la proprietatea tvw1.imagelist
```



```
                .NODES.ADD ("~"+ALLT(STR(codcl)),4,"~"+ALLT(STR(nrfact)) , ;
                        "Fact. nr. " + ALLT(STR(nrfact)) + " din " +DTOC(datafact),4)
            ENDIF

        ENDWITH
    ENDSCAN
ELSE
    MESSAGEBOX("Nu s-au emis facturi!")
ENDIF
SELECT cFactEmise
USE
* se adauga header-ele de coloana la controlul de tip ListView
WITH THISFORM.lvw1
    .SmallIcons = THISFORM.iml1
    WITH .COLUMNHEADERS
        * latimea este stabilita pe o baza conventionala
        .ADD (, , "Nr.", INT(THISFORM.lvw1.WIDTH * 6 / 128))
        * ar fi fost preferabil aliniata la dreapta, dar nu-i permis pentru prima coloana
        .ADD (, , "Denumire produs", INT(THISFORM.lvw1.WIDTH * 40 / 128))
        .ADD (, , "UM", INT(THISFORM.lvw1.WIDTH * 10 / 128))
        .ADD (, , "Cantitate", INT(THISFORM.lvw1.WIDTH * 18 / 128), 1) && alin. dreapta
        .ADD (, , "Pret unitar", INT(THISFORM.lvw1.WIDTH * 16 / 128), 1) && alin. dreapta
        .ADD (, , "Valoare", INT(THISFORM.lvw1.WIDTH * 20 / 128), 1) && alin. dreapta
        .ADD (, , "Val. TVA", INT(THISFORM.lvw1.WIDTH * 18 / 128), 1) && alin. dreapta
    ENDWITH
ENDWITH

* Metoda: populeaza_lista  (adaugata formularului)
LPARAMETERS nrfact_
* se sterge elementele controlului de tip ListView
THISFORM.lvw1.listitems.CLEAR
* se obtin detaliile facturii curente
SELECT linie, denpr, um, cantitate, pretunit, ;
    cantitate*pretunit AS valoare, ;
    cantitate*pretunit*proctva AS valtva ;
    FROM liniifact lf, produse p ;
    WHERE p.codpr=lf.codpr AND lf.nrfact=nrfact_ ;
    INTO CURSOR cLiniiFact ;
    ORDER BY linie

IF _TALLY<>0 && daca s-au prelucrat inregistrari
    SELECT cLiniiFact
    SCAN
    * se adauga elementele listei
        WITH THISFORM.lvw1
            elementcurent = .listitems.ADD(, , ALLT(STR(linie)))
            WITH elementcurent
                .SmallIcon = 1
                .SubItems(1) = denpr
                .SubItems(2) = um
                .SubItems(3) = cantitate
                .SubItems(4) = pretunit
                .SubItems(5) = valoare
                .SubItems(6) = valtva
            ENDWITH
        ENDWITH
    ENDSCAN
    * se adauga o linie de totaluri in lista
    set talk off && altfel se afiseaza totalurile direct pe formular
    * se insumeaza valoarea fara TVA si respectiv valoarea TVA
```

```
    * in doua variabile
    SUM valoare, valtva TO totval, tottva
    set talk on
    * se parcurge colectia SubItems:
    elementcurent = THISFORM.lvw1.listitems.ADD(, , "")
    elementcurent.SubItems(1)="T o t a l"
    elementcurent.SubItems(5)=totval
    elementcurent.SubItems(6)=tottva
    thisform.lblTotal.caption="Total cu TVA: " + allt(str(totval+tottva))
    SELECT cLiniiFact
    USE
ENDIF

                                        *Obiect: tvw1
* Eveniment: NodeClick
*** ActiveX Control Event ***
LPARAMETERS NODE    && acest parametru este transmis de sistem
                                        && si identifica nodul pe care s-a efectuat click
IF VARTYPE(NODE.PARENT)="O" && daca este un nod copil (adica factura, nu client)
    * lanseaza procedura de populare a listei
    * transmitand ca parametru numarul facturii;
    * (e necesar sa se elimine semnul "~" din fata si sa se converteasca
    * in valoare numerica)
    THISFORM.populeaza_lista(VAL(SUBSTR(NODE.KEY,2)))
ELSE                                    && este nod parinte
    * inversam aspectul expandat/comprimat
    NODE.expanded=!NODE.expanded
    WITH THISFORM.lvw1.listitems
        IF .COUNT<>0
            .CLEAR
        ENDIF
    ENDWITH
ENDIF
```

10.5.6. Controlul `Timer`

Controlul `Timer` nu este un control `ActiveX`, ci un control intrinsec Visual FoxPro. Proprietatea cea mai importantă a acestuia, `Interval`, permite specificarea duratei de timp (în milisecunde) la care se declanşează principalul eveniment, `Timer`. Cu toată utilitatea sa în măsurarea timpului, controlul `Timer` nu trebuie folosit drept cronometru, mai ales că nici nu este chiar foarte precis. Astfel, în condiţiile în care sistemul de calcul este solicitat de operaţii mari consumatoare de resurse (calcule complexe, manipularea datelor din tabele de mari dimensiuni, afişarea ecranelor „încărcate" etc.), unitatea centrală va genera evenimente `Timer` la intervale destul de aproximative (în termenii miimilor de secundă, dar nu uitaţi că diferenţele se acumulează în timp). Firește, se poate crea în Visual FoxPro un ceas care să afişeze ora sistemului, dar este indicat ca `Timer`-ul folosit să consulte la intervale regulate ceasul calculatorului, nu să numere secundele de o manieră internă (vezi figura, tabelul şi listingul de mai jos).

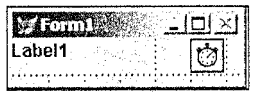


Figura 10.31. Macheta unui ceas simplu

Tabelul 10.6. Proprietăţile controalelor de pe formularul din figura 10.31

Control	*Proprietate*	*Valoare*
`lblTime`	`Caption`	
`Timer1`	`Interval`	500 (1/2 s)
`Timer1`	`Enabled`	`.T.`

Listing 10.14. Codul-sursă pentru formularul de mai sus

```
                                        *Obiect: Timer1
*Eveniment: Timer
IF THISFORM.lblTime.Caption != Time() && daca este afisat altceva decât ora curenta
    THISFORM.lblTime.Caption = Time()
ENDIF
```

În orice caz, un asemenea ceas nu-şi prea dovedeşte utilitatea, de vreme ce dispunem de comanda `SET CLOCK...` Totuşi, controlul `Timer` s-ar putea folosi în operaţii de cronometrare a unor operaţii de durată (cu afişarea timpului scurs) – unde funcţionalitatea sa ar putea fi combinată cu aceea a controlului `ProgressBar` – ori în declanşarea periodică a unor operaţii de actualizare, în condiţiile accesului multiutilizator. O altă aplicaţie tipică ar consta în execuţia unei operaţii la un moment prestabilit, în condiţiile absenţei operatorului. De remarcat că „pornirea", respectiv „oprirea" `Timer`-ului au loc prin atribuirea valorilor `.T.` sau `.F.` proprietăţii `Enabled`.

Limita maximă a proprietăţii `Interval` este `2147483647` (printr-un calcul simplu, obţinem circa 596,52 ore, adică aproape 25 de zile). Peste această durată e nevoie de un artificiu de programare, în care timpul scurs să se acumuleze într-o variabilă şi aceasta să fie folosită ca element de referinţă.

La modul cel mai simplu, un control `Timer` ar putea determina afişarea intermitentă a unui mesaj de atenţionare, prin manipularea proprietăţii `Visible` (alternând valorile `.T.` şi `.F.`) a unui control `Label` ori `TextBox`.

Într-o posibilă aplicaţie destinată pieţei financiare, un control `Timer` ar putea lansa, la fiecare *n* minute, o procedură pentru afişarea cursurilor diferitelor devize sau acţiuni, a volumului tranzacţiilor încheiate până la momentul respectiv etc.

Să considerăm un scenariu în care o aplicaţie este exploatată în reţea de mai mulţi utilizatori. După cum se va vedea în capitolul 13, accesul multiutilizator presupune *blocarea înregistrărilor*. În cursul blocării, respectivele înregistrări nu sunt accesibile altor utilzatori pentru modificare. Ca urmare, am putea folosi controlul `Timer` pe de o parte pentru a atenţiona operatorul că blocarea iniţiată de el durează mai mult de un timp prestabilit ca rezonabil, iar pe de altă parte, pentru a încerca automat, la intervale de timp predefinite, comiterea unor tranzacţii împiedicate iniţial ca urmare a blocării unor înregistrări de către alţii.

10.6. Depanarea formularelor cu utilitarul Debugger

În capitolul 4 am prezentat posibilitățile de monitorizare a execuției unui program (`.prg`) în vederea depistării și corecției erorilor. În cazul formularelor însă, codul este împărțit între procedurile diverselor evenimente, a căror ordine de apariție nu este întotdeauna evidentă, mai ales dacă în mod deliberat programatorul apelează o metodă a unui obiect din procedura altui eveniment al acelui obiect sau al altuia. Secvențele de cod cuprinse în formulare complexe, mai ales dacă sunt documentate în grabă sau deloc, devin cu timpul o adevărată „junglă" în care până la urmă chiar autorul lor întâmpină dificultăți în a surprinde erorile subtile. Pentru astfel de cazuri, instrumentul `Debugger` vine în ajutorul dezvoltatorilor cu opțiunea *Toggle Event Logging*, ce permite jurnalizarea apariției evenimentelor în fereastra `Debug Output` sau scrierea lor într-un fișier `.log`, pentru analize ulterioare.

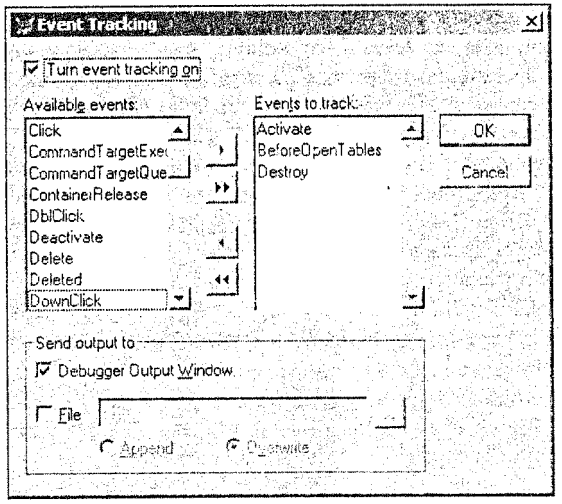


Figura 10.32. Opțiunea pentru monitorizarea evenimentelor

La activarea acestei opțiuni (cu butonul de pe bara utilitarului `Debugger`) se deschide fereastra din figura 10.32, în care se selectează evenimentele a căror apariție se dorește a fi semnalată. Evenimentele adăugate listei din dreapta rămân permanent acolo; monitorizarea se pornește sau oprește pentru toate evenimentele cu ajutorul căsuței de opțiune intitulate *Turn Event Tracking on*. După închiderea dialogului cu `OK`, poziția apăsată a butonului arată că opțiunea este efectivă. Ulterior acestei operații, se încarcă formularul în `Debugger` prin acționarea butonului , pe ecran afișându-se dialogul din figura 10.33. Se alege formularul dorit, acțiunea încheindu-se cu un click pe butonul `Do`. În continuare, pentru execuția formularului se mai acționează o dată butonul , după care se procedează în mod normal, cu posibilitatea comutării de la fereastra principală Visual FoxPro la fereastra `Debugger`. Cât despre evenimentele propuse anterior spre monitorizare, acestea vor fi marcate în fereastra `Debug Output`, în ordinea în care apar.

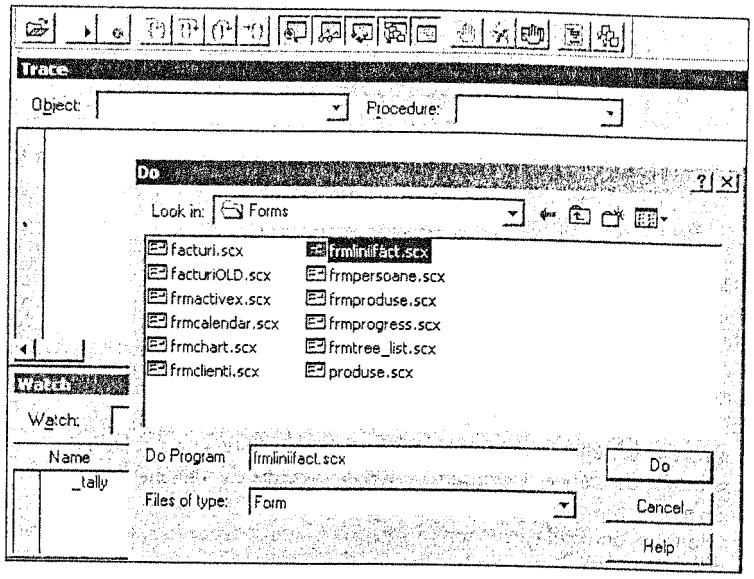


Figura 10.33. Încărcarea unui formular în `Debugger`

În figura 10.34 s-a prezentat o parte a ferestrei `Debug Output` (care a fost în prealabil maximizată pentru o vedere mai bună), în care se observă declanșarea evenimentului `Destroy` cu ocazia închiderii formularului (întâi pentru formular, apoi pentru fiecare obiect de pe formular în parte).

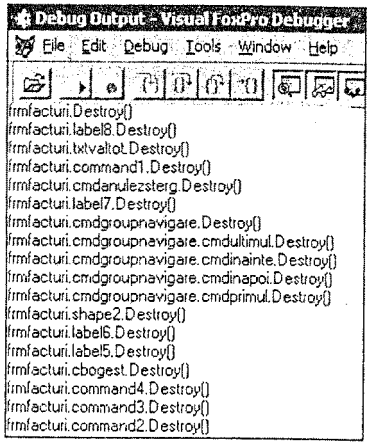


Figura 10.34. Evenimente monitorizate în fereastra `Debug Output`

Capitolul 11

Programare orientată-obiect în Visual FoxPro

Considerată una dintre tehnicile moderne de programare, programarea orientată-obiect este bazată pe *paradigma obiect-mesaj*. Fundamentul acesteia este constituit din *obiecte*; fiecare obiect răspunde la o colecție determinată de *mesaje*. În programarea procedurală, subrutinele sau funcțiile se apelează unele pe altele pentru a returna anumite date sau pentru a modifica anumiți parametri de intrare-ieșire. În modelul obiect-mesaj, în loc de date de tip scalar (număr, șir de caractere etc.) sau vectori ori matrice de date dispunem de un *obiect* capabil să proceseze cereri cunoscute ca *mesaje*. Aceste mesaje:

- pot cere obiectului să efectueze „ceva" și să returneze un rezultat;
- pot modifica starea obiectului (adică pot modifica valorile unor proprietăți ale obiectului).

În acest context, obiectul este o combinație între date și procedurile sau funcțiile folosite la manipularea lor. Programul informatic devine în atare condiții „un grup de obiecte care-și spun unul altuia ce să facă"[52].

11.1. Delimitări conceptuale

Programarea orientată-obiect a pornit de la dezvoltarea conceptului de *tip abstract de dată* (TAD). Multe limbaje (procedurale, ca de exemplu Basic, C sau Pascal) dau posibilitatea utilizării unor tipuri de date compozite, definite de utilizator (*user-defined types*). Datele din aceste tipuri se manipulează ca un tot unitar, în același timp fiind posibilă și accesarea „cărămizilor" acestui „tot", ce sunt date de tipuri elementare. De exemplu, în BASIC putem crea un tip de dată definit de utilizator cu instrucțiunea *TYPE...END TYPE*, iar în C și C++ cu instrucțiunea *struct*.

Listing 11.1. Crearea și utilizarea unui tip de dată definit de utilizator în Basic...

```
' aceasta este o declaratie de tip de data definit de utilizator in BASIC
Private Type PC
  Procesor As Variant
  Memorie As Long
  Video As Integer
  Pret As Currency
  DataCumpararii As Variant
End Type
'utilizarea respectivului tip de data
'acum declaram o data de tipul definit mai sus
Dim CalculatorulMeu as PC
CalculatorulMeu.Procesor="Pentium III"
```

52. Frază folosită pentru caracterizarea limbajului Smalltalk, un pionier al limbajelor orientate pe obiecte, de către unul dintre „părinții" săi, Alan Kay.

Listing 11.2. ...și în C/C++

```
//aceasta este declaratia de tip de data definit de utilizator in C/C++
struct PC
{char Procesor[10];
long Memorie;
int Video;
float Pret;
char DataCumpararii[10];
};
//utilizarea tipului abstract de dată
//acum declaram doua date de tipul definit mai sus
struct PC CalculatorulMeu;          //declaratie in stil C
PC CalculatorulTau;                 //declaratie in stil C++
//initializare (preturile sunt exprimate in dolari)
CalculatorulMeu.Pret=2250.99;
CalculatorulTau.Pret=1995;
```

Visual FoxPro nu dă posibilitatea definirii directe a unui asemenea tip de dată. Acesta se poate simula, de pildă, printr-un tablou cu două dimensiuni sau prin înregistrarea curentă a unei tabele, în care fiecare câmp are propriul său tip de dată, dar înregistrarea în ansamblu ar constitui un tip nou de dată cu o structură compusă. Crearea unei noi date de un asemenea tip s-ar face prin adăugarea de linii în respectiva tabelă. Ce-i drept, nu se poate da un nume înregistrării; este însă posibil să avem un câmp special care să rețină o valoare ce identifică unic înregistrarea – cu alte cuvinte, o cheie primară.

Listing 11.3. O variantă Visual FoxPro

```
*crearea unei tabele temporare
CREATE CURSOR Calculatoare(Nume c(20), Procesor c(10), ;
    Memorie n(6), Video n(6), Pret n(14,2), DataCumpărării D)

*acum declaram doua date de tipul definit mai sus
INSERT INTO Calculatoare VALUES ("Calculatorul meu", "Pentium III", 65536, 8192, ;
        2122, {02/03/2000})
INSERT INTO Calculatoare VALUES ("Calculatorul tau", "Athlon",  65536, 16484, ;
        1850, {02/12/2000})
SELECT Calculatoare
LOCATE FOR LOWER(nume)="calculatorul meu"
IF FOUND()
    ? "Calculatorul meu a costat" + Calculatoare.Pret +" USD."
END IF
```

La un moment dat, în teoria programării s-a ivit ideea perfecționării tipurilor abstracte de date, care să înglobeze, pe lângă *valori ale variabilelor* (care redau o stare la un moment dat), și procedee de descriere a *comportamentului* și să ofere o *interfață* pentru accesarea *stării* și pentru interpretarea/declanșarea *comportamentului*. Adăugând unor date și operațiile posibil de realizat asupra lor, modelul informatic devine mai apropiat de modelul lumii reale, în care la tot pasul avem de-a face cu diferite *obiecte*. Firește, complexitatea lumii reale e greu de simulat printr-un program informatic și se pune chiar problema în ce măsură are loc transpunerea acestei complexități în produsul-program. Cum majoritatea aplicațiilor sunt orientate către o problemă anume, *abstractizarea* devine o soluție salvatoare. În loc de a memora toate detaliile nesemnificative (mai bine zis, inutile pentru modelarea/rezolvarea problemei), abstractizarea permite ignorarea lor și concentrarea pe

aspectele strict necesare atingerii scopului urmărit. De exemplu, atunci când vă aşezaţi în faţa calculatorului, vă este indiferent cum funcţionează diferitele părţi electronice şi mecanice care-l compun, cum se memorează datele pe discul magnetic, ce semnal se generează când efectuaţi un click de mouse etc. Aceasta este preocuparea constructorilor calculatorului. De asemenea, dacă nu sunteţi pasionat(ă) de programare, nu se pune problema de a şti cu ce instrucţiuni de nivel scăzut se face minimizarea unei ferestre ori afişarea unei casete de dialog. Dumneavoastră, ca utilizator final, folosiţi dispozitivele periferice standard şi lucraţi cu câteva aplicaţii care arată şi se comportă aproape identic cu omoloagele lor de pe calculatorul unui coleg, chiar dacă acel calculator este mai mult sau mai puţin diferit constructiv.

Abstractizarea dă posibilitatea grupării obiectelor cu caracteristici şi comportamente similare în *clase*. În programarea orientată-obiect, „clasele" şi „obiectele" sunt două concepte foarte importante, dar nu trebuie confundate. Astfel, o clasă reprezintă o definiţie a modului cum trebuie să fie construit un obiect şi cum trebuie să se comporte el. Obiectul reprezintă o entitate creată pe baza sus-numitei definiţii. Figura 11.1 şi explicaţiile aferente vor ajuta la spulberarea, măcar parţială, a aurei de mister din jurul conceptului de „clasă" şi al noţiunilor asociate.

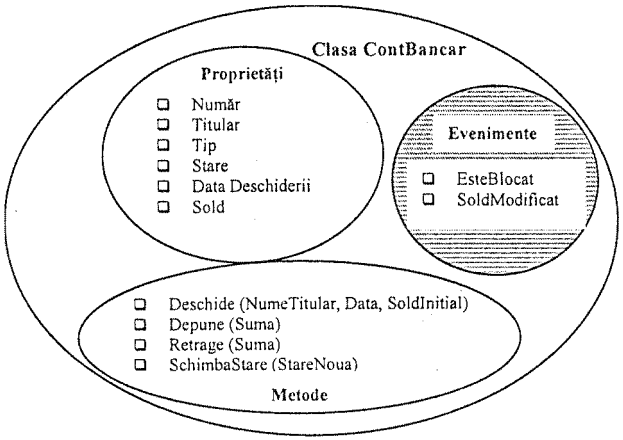


Figura 11.1. O schiţă a clasei `ContBancar`

Astfel, o definiţie minimală a clasei ar cuprinde descrierea *proprietăţilor* şi *metodelor* acestei clase. Proprietăţile (sau *atributele*, ori *variabilele-membru*) sunt acele trăsături ale obiectului care au fost reţinute ca importante în procesul de abstractizare. Metodele (sau *operaţiile*, ori *funcţiile-membru*) sunt secvenţe de cod care definesc comportamentul tuturor obiectelor dintr-o clasă. Ele sunt legate în mod inextricabil de clasa respectivă, neputând fi folosite independent, ca procedurile „normale". Teoria ne învaţă că o metodă se declanşează ca urmare a primirii unui *mesaj* de către obiectul care o deţine. Mesajul poate veni de la alt obiect, fiind produs de o metodă a acestuia, ori poate veni de la sistemul de calcul, care codifică într-un mod anume o acţiune a utilizatorului sau o cerere a sistemului de operare. În acest din urmă caz intră în scenă *evenimentele*, specifice programării în mediile Windows şi similare.

Numele proprietăţilor şi *semnăturile* metodelor (adică numele lor şi ale eventualilor parametri de intrare-ieşire ai fiecăreia) formează *interfaţa* unei clase, care este publică.

Codul-sursă asociat metodelor formează *implementarea*, care este inaccesibilă utilizatorului[53]. Respectivul cod poate fi modificat doar prin intervenţie în definiţia clasei.

Împachetarea într-o singură entitate a codului care defineşte proprietăţile şi metodele se numeşte *încapsulare* şi este unul dintre atuurile modelului de programare obiectuală. Faptul că un program „utilizator" are acces numai la acele proprietăţi şi metode pentru care s-a dorit explicit aceasta la momentul programării clasei, combinat cu imposibilitatea modificării codului metodelor din afara clasei respective, defineşte un alt concept cunoscut ca *ascunderea informaţiei*.

Mecanismul prin care se creează obiecte pornind de la definiţia (şablonul) încorporat în clasă poartă numele de *instanţiere*. Ca urmare, un obiect se mai numeşte şi *instanţă* a unei clase. Dacă definiţia clasei este persistentă, fiind stocată într-un fişier, obiectele create sunt temporare, existenţa lor luând sfârşit la terminarea programului creator.

Obiectele provenind din aceeaşi clasă diferă între ele prin valorile diferite ale proprietăţilor lor[54], dar partajează codul care formează implementarea. De aceea, cu cât este nevoie de mai multe obiecte de un anumit fel, cu atât mai avantajoasă se dovedeşte folosirea clasei. Cu alte cuvinte, clasele permit *reutilizarea codului*. Abstractizarea se dovedeşte prezentă şi aici: ca să folosim o clasă, ne interesează mai puţin *cum* se formează şi populează în memorie diferite structuri de date reprezentând obiectele sale (aceasta e misiunea compilatorului limbajului, respectiv a bibliotecilor *runtime* ale sale), decât *ce* sunt capabile să facă acele obiecte.

Clasele sunt o extensie a tipurilor de date definite de utilizator; posibilitatea definirii claselor în Visual FoxPro rezolvă problema inexistenţei unei instrucţiuni pentru definirea de astfel de tipuri: putem pur şi simplu defini o clasă care are numai proprietăţi, fără metode.

Obiectele din lumea reală tind să fie organizate. De fapt nu tind ele singure, ci oamenii au plăcerea sau pasiunea de a clasifica, ordona, ierarhiza totul. Ierarhizarea intervine atunci când la perceperea unor obiecte intervin niveluri de abstractizare diferite din ce în ce mai înalte. Să luăm un exemplu aflat la îndemână: serviciul de evidenţa populaţiei are nevoie să cunoască următoarele proprietăţi ale unei persoane: numele, prenumele, data naşterii, numele tatălui, numele mamei, starea civilă. O facultate la care este înscrisă o persoană are nevoie de toate detaliile de mai sus, plus: anul înscrierii, anul de studii, specializarea, grupa etc. O firmă la care lucrează respectiva persoană cere toate detaliile de la evidenţa populaţiei, plus: data angajării, funcţia, secţia, salariul de bază etc. Dar atât entitatea *student*, cât şi entitatea *salariat* au la bază entitatea *persoană*. Nu ar fi util ca, la crearea entităţii *student* să „împrumutăm" de la entitatea *persoană* toate elementele deja existente, adăugând la ele anul înscrierii, anul de studii, specializarea, grupa...? Un raţionament similar se aplică entităţii *salariat*. Un asemenea fenomen de propagare „în jos" a funcţionalităţii unei clase se numeşte în teoria programării obiectuale *moştenire*. Pentru înţelegerea facilă a conceptului, raţionamentul de mai sus a fost simplificat doar la nivelul proprietăţilor, dar moştenirea se aplică în egală măsură şi metodelor. Crearea unei *clase derivate* sau *subclase* permite o scădere drastică a cantităţii de cod ce trebuie scrisă, prin faptul că se va adăuga cod doar pentru noile proprietăţi şi metode. Clasele *student* şi *salariat* se vor numi *clase derivate*. Întreţinerea ulterioară a aplicaţiei care foloseşte una sau

53. Prin *utilizator* nu se înţelege aici, *stricto sensu*, persoana care foloseşte programul. Poate fi vorba de programatorul unei aplicaţii care include clase definite în prealabil sau, prin extensie, de un program „creator" ori „consumator" de obiecte.

54. Dacă două obiecte au exact aceleaşi valori ale proprietăţilor, nu înseamnă că ele sunt identice. Sistemul diferenţiază obiectele prin identificatori interni, unici (la modul general, se vorbeşte de OID – *Object IDentifier*).

mai multe dintre aceste trei clase devine mai simplă. Orice modificare a funcționalității în clasa de bază este automat propagată în clasele derivate. Revenind la exemplul de mai sus, dacă evidența populației dorește și memorarea situației serviciului militar, atunci se adaugă un atribut *serviciu_militar* în clasa *persoană*, iar clasele derivate *student* și *salariat* vor avea automat acces la acest atribut.

Dacă la crearea unei clase se dorește explicit ca o proprietate sau metodă să nu poată fi „vizibilă" în afara clasei respective, se pot folosi *modificatorii de acces*: respectiva proprietate poate fi declarată „privată" ori „protejată". Formatul acestor specificatori diferă de la un limbaj la altul. În Visual FoxPro ei sunt:

- `PUBLIC` – semnifică faptul că proprietatea sau metoda declarată astfel poate fi accesată din orice porțiune de cod (din clasă sau din afara definiției ei); este modificatorul de acces implicit și se poate omite;
- `PROTECTED` – proprietatea (sau metoda) declarată astfel poate fi accesată din codul clasei în care se găsește ori din codul-sursă al unei clase derivate (subclase);
- `HIDDEN` – proprietatea (sau metoda) declarată astfel poate fi accesată numai din codul clasei în care se găsește.

Polimorfismul este un concept înrudit cu moștenirea și se referă, într-o definiție consacrată, la „posibilitatea de a transmite același mesaj diferitelor obiecte, care vor executa acțiuni diferite". Pentru a fi ceva mai pe înțelesul cititorului nefamiliarizat cu literatura de specialitate, vom spune că polimorfismul este capacitatea ca două clase distincte să aibă câte o metodă cu nume identic, dar cu efect diferit (de exemplu, metoda `Refresh` este întâlnită în clasele `Form` și `Grid`, dar nu lucrează chiar la fel în ambele).

11.2. Clase în Visual FoxPro

Pe lângă un bogat sortiment de clase predefinite, Visual FoxPro dispune de facilități ce oferă programatorului posibilități extrem de vaste de creare și utilizare a propriilor clase. Moștenirea și subclasarea sunt suportate atât de clasele de bază Visual FoxPro, cât și de clasele definite de utilizator.

11.2.1. De ce să folosim clase?

Clasele pot fi utilizate în mai multe situații complexe. O proiectare judicioasă a aplicației va prevedea ce clase să includem și ce funcționalitate să conferim fiecăreia. Respectiva funcționalitate trebuie să asigure o adecvare cât mai bună a codului la problemele avute în vedere. Nu întâmplător se apreciază că, în dezvoltarea unei aplicații în tehnica orientată-obiect, majoritatea timpului este alocat etapelor de analiză și proiectare.

11.2.2. Clasele de bază Visual FoxPro

Cu clasele de bază Visual FoxPro v-ați întâlnit pe parcursul capitolelor privitoare la formulare: orice formular ori control este de fapt un obiect creat ad-hoc (la „desenarea" sa), pe baza unei definiții memorate în bibliotecile Visual FoxPro.

Tabelul 11.1. Clasele de bază Visual FoxPro

`ActiveDoc`	`Custom`	`Label`	`PageFrame`
`CheckBox`	`EditBox`	`Line`	`ProjectHook`
`Column*`	`Form`	`ListBox`	`Separator`
`CommandButton`	`FormSet`	`OLEBoundControl`	`Shape`
`CommandGroup`	`Grid`	`OLEContainerControl`	`Spinner`
`Combo-Box`	`Header*`	`OptionButton*`	`TextBox`
`Container`	`Hyperlink Object`	`OptionGroup`	`Timer`
`Control`	`Image`	`Page*`	`ToolBar`

* aceste controale sunt membri ai unor clase-container și nu pot fi subclasate.

Recunoașteți în tabelul 11.1 obiectele de tip *control* și de tip *container* despre care s-a discutat la începutul capitolului 8. Aceste clase sunt organizate într-o ierarhie[55] pe care o prezentăm în figura 11.2. Clasele de tip control sunt mai complete decât clasele de tip container, dar sunt mai puțin flexibile.

Orice clasă de bază Visual FoxPro recunoaște un set minim de evenimente și un set minim de proprietăți pe care le redăm în tabelele 11.2 și 11.3.

Tabelul 11.2. Evenimentele comune tuturor claselor de bază Visual FoxPro

Eveniment	*Descriere*
`Init`	Apare la crearea obiectului
`Destroy`	Apare la distrugerea obiectului
`Error`	Apare când survine o eroare în procedurile interne clasei (metode ori proceduri-eveniment)

Tabelul 11.3. Proprietățile comune tuturor claselor de bază Visual FoxPro

Proprietate	*Descriere*
`Class`	Clasa de care aparține obiectul
`BaseClass`	Clasa de bază, dacă clasa curentă este derivată (clasa de bază este cea de la „rădăcina" ierarhiei de clase)
`ClassLibrary`	Numele bibliotecii* în care este memorată definiția clasei
`ParentClass`	Clasa-părinte, dacă clasa curentă este derivată (dacă este derivată direct dintr-o clasă de bază Visual FoxPro, `ParentClass` are aceeași valoare cu `BaseClass`)

* un fișier-bibliotecă poate conține definițiile mai multor clase și are extensia `.VCX`.

11.2.3. Clasele definite de utilizator

Programatorul unei aplicații poate decide să creeze câte o clasă pentru fiecare control au formular particular pe care dorește să-l utilizeze în respectiva aplicație – dar este oare ceasta cea mai bună cale? Două sunt practic consecințele unei asemenea decizii:

- vom obține o redundanță nedorită, dată de multe clase ale căror misiuni se suprapun mai mult sau mai puțin, dar care trebuie întreținute separat (deci nu ne vom îndepărta prea mult de situația inițială a folosirii controalelor intrinsece, la care

[55] Porțiuni ale acestui material au la bază subcapitolul „Object-Oriented Programming" din *Visual FoxPro Programmer's Guide*, versiunea electronică (MSDN).

orice modificare – să zicem, modificarea culorii textului – trebuie reluată de *n* ori, unde *n* este numărul de controale similare);

- vom găsi pentru o clasă prea puține utilizări în aplicația respectivă și poate și mai puține într-o altă aplicație (deci gradul de reutilizare a software-ului, care este unul dintre mobilurile programării obiectuale, va fi redus).

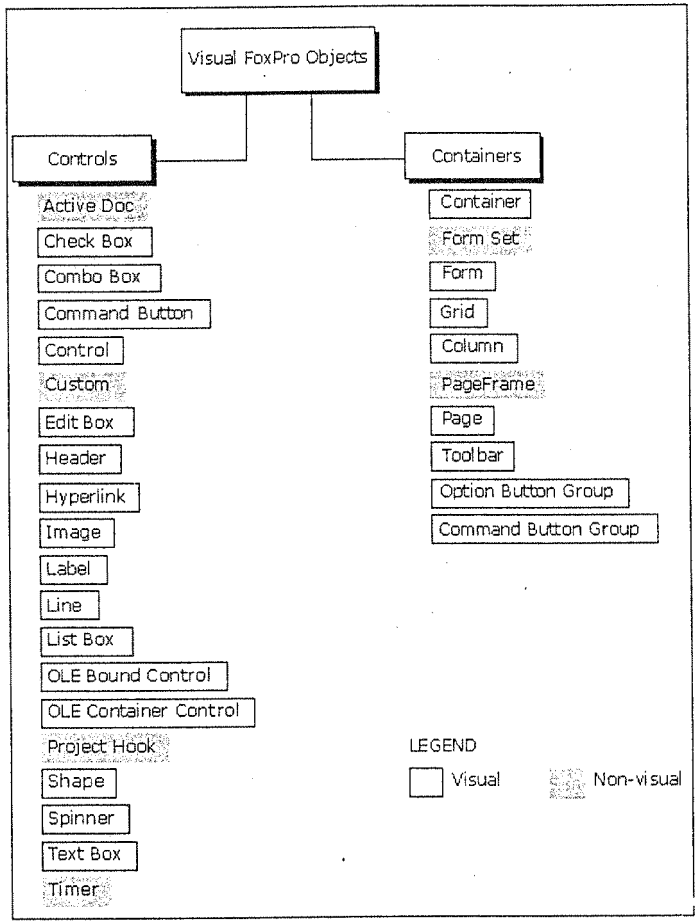


Figura 11.2. Ierarhia de clase Visual FoxPro

Este drept, programatorul poate avea satisfacția de a fi reușit o primă aplicație în care aproape totul a fost „orientat-obiect". Reamintiți-vă însă că nimeni nu face programare „de dragul programării", dacă dorește să obțină o aplicație cu adevărat fiabilă.

În principal, utilitatea claselor în Visual FoxPro se referă la:

- încapsularea unei funcționalități generice (de exemplu, butoane care permit navigarea printre înregistrările oricărei tabele `.dbf` ori etichete care-și schimbă culoarea la trecerea mouse-ului pe deasupra lor);

- oferirea unui aspect (*look-and-feel*) consistent al interfeței (de exemplu, folosirea unei casete de text cu efecte de umbrire ori crearea unor butoane cu aspect neobișnuit, care să fie folosite peste tot în aplicație, în locul controalelor intrinsece Visual FoxPro).

Cât despre felurile de clase ce pot fi create, programatorul are la dispoziție mai multe opțiuni:

- subclasarea claselor vizuale (afișabile) de bază Visual FoxPro (un exemplu tipic îl constituie un buton care să permită părăsirea oricărui formular la apăsarea sa);
- crearea de noi clase de tip container, care să conțină derivate ale mai multor clase de bază Visual FoxPro (de exemplu, o căsuță de text însoțită automat de o etichetă la trasarea sa pe formular);
- crearea de clase non-vizuale (destinate unor calcule sau prelucrări simbolice, cum ar fi numărarea cuvintelor dintr-un control de tip `EditBox`, ori transcrierea în litere a unui număr, ori conversii din sistemul anglo-saxon în cel metric etc.).

Crearea de noi clase poate avea loc în mod vizual, folosind instrumentul `Class Designer`, sau în mod programatic (prin cod). Nu trebuie confundată maniera de proiectare cu aspectul vizual sau non-vizual, la execuție, al obiectelor aparținând de aceste clase.

11.2.4. Crearea de clase cu instrumentul `Class Designer`

Class Designer este asemănător întru câtva cu *Form Designer*, oferind suprafețe pe care se pot plasa controale (dacă se dorește crearea unei clase vizuale), dar și setul predefinit de evenimente Visual FoxPro, care se pot interpreta. În plus, clasei i se pot adăuga proprietăți și metode specifice. Se lansează fie din fereastra de comenzi, prin comanda:

```
CREATE CLASS [<nume-clasă> OF <nume-bibliotecă> [AS <nume-
clasă-părinte> [FROM <nume-bibliotecă-părinte>]]],
```

fie, mai comod, din meniul `File→New...` ori cu butonul 🗋, alegând opțiunea `Class`. De asemenea, dacă lucrăm cu un proiect deschis, se poate folosi butonul `New...`, după ce se alege secțiunea `Libs` a proiectului. Va apărea un dialog similar celui din figura 11.3.

Subclasarea unui control existent. Clasele `CmdBtnQuit` și `CmdBtnColorQuit`

Vom demonstra aici crearea unui buton care să producă părăsirea unui formular la apăsarea sa. Din moment ce acest buton va fi întotdeauna plasat pe un formular, rezultă că evenimentul `Click` al său va trebui să conțină doar instrucțiunea `thisform.release`.

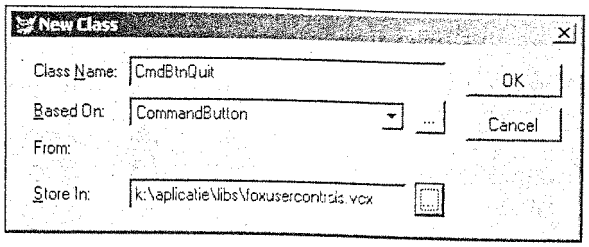


Figura 11.3. Dialogul de creare a unei noi clase Visual FoxPro

Vom lansa dialogul de creare a unei noi clase. În figura 11.3 s-au specificat, de sus în jos, numele noii clase: `CmdBtnQuit` (se scrie de la tastatură), numele clasei-părinte `CommandButton` (se alege din listă, iar butonul cu ... din dreapta listei permite încărcarea unei alte biblioteci decât cea implicită), calea şi numele fişierului-bibliotecă în care vom memora definiţia prezentei clase (se poate alege o bibliotecă existentă cu ajutorul butonului ...).

După încheierea dialogului, cu butonul OK, se afişează fereastra `Class Designer`, care în cazul de faţă (figura 11.4) conţine imaginea unui buton (datorită alegerii făcute pentru clasa-părinte).

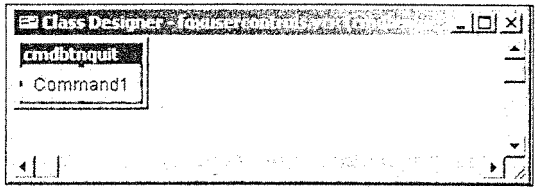


Figura 11.4. Fereastra `Class Designer`

În continuare, vom schimba valoarea proprietăţii `Caption` a butonului în "E\<xit" (deci va putea fi acţionat şi cu combinaţia `Alt+X`), iar în procedura evenimentului `Click` vom scrie doar `thisform.release`. Închiderea ferestrei `Class Designer` are drept consecinţă afişarea dialogului de salvare a definiţiei clasei respective (dacă nu s-a salvat între timp). Clasa este gata, dar instanţierea sa nu poate avea loc decât prin desenarea unui control de tip `CmdBtnQuit`. În acest scop, vom deschide unul din formularele create anterior, să zicem `frmChart`. Pentru a avea acces la clasa `CmdBtnQuit`, trebuie să aducem pe bara de instrumente `Form Controls` elementele altei biblioteci decât biblioteca Visual FoxPro standard (ce conţine controalele intrinsece), iar acest lucru este posibil dacă se procedează ca în figura 11.5. Bara de instrumente se va modifica, având aspectul din figura 11.6 (în mod asemănător s-a procedat pentru controalele `ActiveX`, în capitolul 10). În continuare, alegând noul buton din bara de instrumente, se va trasa pe formularul `frmChart` un control din clasa `CmdBtnQuit`.

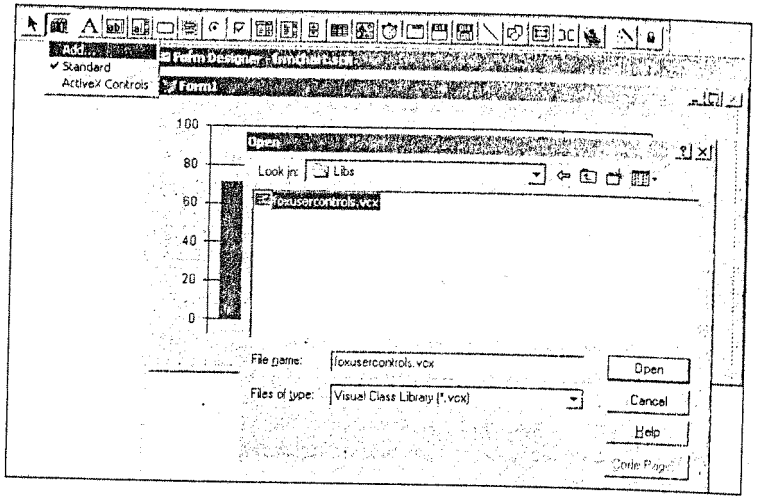


Figura 11.5. Adăugarea de clase definite de utilizator

Vom constata că:

- numele controlului este `Cmdbtnquit1` (Visual FoxPro denumeşte automat instanţele unei clase cu numele acesteia plus numărul instanţei), dar se poate modifica în fereastra de proprietăţi;
- textul afişat pe acesta este `Exit` (stabilit la proiectarea clasei), dar se poate modifica în fereastra de proprietăţi;
- după lansarea în execuţie a formularului şi efectuarea unui click pe butonul `Cmdbtnquit1`, are loc dispariţia acestuia de pe ecran.

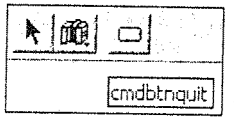


Figura 11.6. Aspectul barei de instrumente după adăugarea clasei `CmdBtnQuit`

Notă. Dacă ulterior este nevoie pe formular de controalele intrinsece Visual FoxPro, se poate reveni la bara de instrumente obişnuită printr-un nou click pe butonul, urmat de alegerea numelui bibliotecii, *Standard*.

Aţi remarcat, la acţionarea butonului `Cmdbtnquit1`, dispariţia formularului de pe ecran, *deşi în evenimentul* `Click` *al acestui buton nu există nici o linie de cod.* Este vorba de *apelul codului metodelor într-o ierarhie de clase.* Visual FoxPro implementează moştenirea metodelor, astfel încât, la apariţia unui eveniment al unui obiect:

- se execută doar codul scris în metoda respectivă de la nivelul obiectului în cauză (se spune că metoda este suprascrisă);
- se execută automat codul metodei specifice din clasa de care ţine obiectul, dacă nu există cod pentru metodă la nivelul obiectului însuşi. Dacă nu există cod în clasa respectivă, dar există într-o clasă-părinte a acesteia (bineînţeles, pentru acelaşi

eveniment), se va executa cel din clasa-părinte, şi tot aşa. Evident, o clasă derivată poate suprascrie o metodă a clasei sale părinte.

Dacă se doreşte invocarea explicită a codului metodei clasei-părinte atunci când subclasa are cod pentru respectiva metodă, trebuie inclus în codul din clasa derivată apelul funcţiei `DODEFAULT()`. Pentru a ilustra acest exemplu, în metoda `Click` a obiectului `Cmdbtnquit1` de pe formularul `frmChart` se scrie codul din listingul 11.4, având ca menire consultarea utilizatorului asupra intenţiei de părăsire a formularului[56].

Listing 11.4. Codul evenimentului `Click` al butonului `cmdbtnquit1`

```
# INCLUDE "\aplicatie\constantemsgbox.txt"
IF IDNO=MESSAGEBOX("Abandonati acest grafic?", ;
        MB_YESNO+MB_ICONQUESTION,"Mesaj")
    RETURN
ENDIF
```

Notă. Fişierul `constantemsgbox.txt` conţine declaraţii de constante (`#DEFINE`) pentru numerele care definesc stilul casetei de dialog, precum şi pentru răspunsurile posibile. Nu este listat aici.

La o nouă execuţie a formularului, vom constata că la apăsarea butonului nu se întâmplă nimic, chiar dacă răspundem `Yes`, întrucât codul din listingul 11.4 suprascrie codul metodei `Click` a acestei clase. Corect ar fi ca acest cod să se prezinte ca în listingul 11.5, ori, mai elegant încă, sub forma din listingul 11.6.

Listing 11.5. Apelarea selectivă a unei metode în susul ierarhiei de clasă

```
#INCLUDE "\aplicatie\constantemsgbox.txt"
IF IDNO=MESSAGEBOX("Abandonati acest grafic?", ;
        MB_YESNO+MB_ICONQUESTION,"Mesaj")
    RETURN
ELSE
    DODEFAULT()
ENDIF
```

Listing 11.6. Variantă a codului din listingul 11.5

```
#INCLUDE "\aplicatie\constantemsgbox.txt"
IF IDYES=MESSAGEBOX("Abandonati acest grafic?", ;
        MB_YESNO+MB_ICONQUESTION,"Mesaj")
    DODEFAULT()
ENDIF
```

De data aceasta butonul funcţionează corect. Observaţi că nu se mai scrie nicăieri `thisform.release`, această operaţie fiind asigurată de codul metodei `Click` din clasa `CmdBtnQuit`, cod apelat prin funcţia `DODEFAULT()`.

Dacă într-o metodă oarecare se doreşte apelul unei alte metode, din clasa de bază sau din altă clasă, atunci trebuie să folosim o sintaxă ce implică *operatorul de rezoluţie*, anume „`::`". Reluând exemplul din listingul 11.6, `DODEFAULT()` şi `CmdBtnQuit::Click` produc rezultate similare. Dar operatorul `::` îşi găseşte utilitatea în „sărirea" peste treptele ierarhiei de clasă, în stânga sa putându-se scrie numele unei clase-părinte, ori „bunic", ori „străbunic", după necesităţi şi după priceperea programatorului.

56. Termenul corect pentru butonul `Cmdbtnquit1` este de fapt acela de „instanţă a clasei `CmdBtnQuit`", nu de „clasă derivată", dar moştenirea este valabilă şi pentru instanţele unei clase, deci rolul funcţiei `DODEFAULT()` se păstrează.

Pentru a întări cele spuse, vom deriva în continuare, din clasa `CmdBtnExit`, clasa `CmdBtnColorExit`, a cărei trăsătură notabilă este aceea că, la poziţionarea mouse-ului deasupra sa, culoarea textului va vira către una stabilită de programator la proiectarea clasei. Vom numi această „culoare de schimb" cu numele `AlternateForeColor`, constituind-o ca proprietate distinctă a clasei.

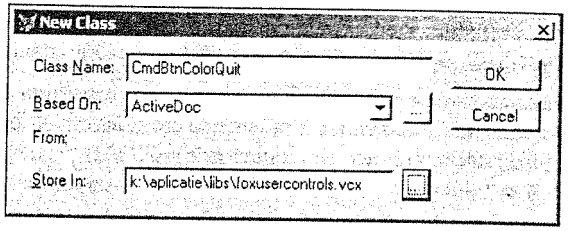


Figura 11.7. Crearea unei noi clase

Dialogul de creare a clasei va fi cel din figura 11.7 (vom memora definiţia clasei în acelaşi fişier-bibliotecă `foxusercontrols.vcx`, ca şi pentru precedenta clasă). Nu vom termina dialogul doar după denumirea clasei şi selectarea fişierului-bibliotecă. Mai este nevoie de alegerea clasei-părinte. Evident, nu vom lucra cu `ActiveDoc` din figura 11.7, nici cu alta dintre clasele intrinsece Visual FoxPro, ci cu clasa `CmdBtnQuit`, din biblioteca `foxusercontrols`. Selecţia acestei clase-părinte are loc folosind butonul notat cu ... de lângă lista `Based On`..., consecinţa fiind apariţia dialogului din figura 11.8.

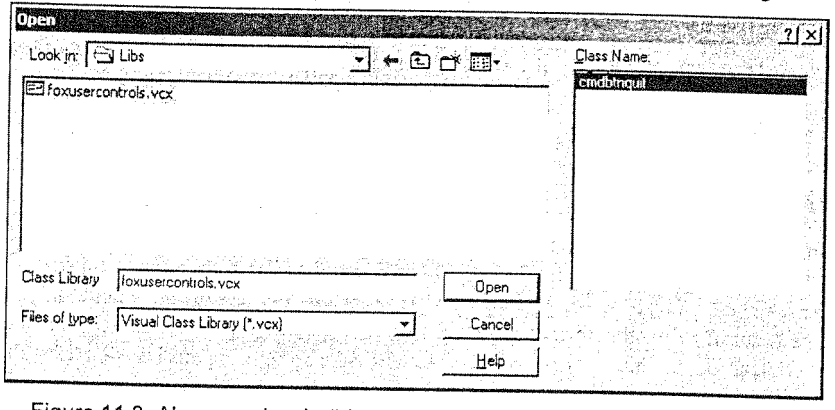


Figura 11.8. Alegerea clasei-părinte, alta decât o clasă de bază Visual FoxPro

După apăsarea pe butonul `Open`, vom dispune de un „banc de lucru" similar celui de la clasa anterioară (figura 11.4).

Vom adăuga (`Class→New Property...`) proprietatea `AlternateForeColor` (vezi figura 11.9), apoi vom completa metoda `MouseMove`, imediat după declaraţia implicită `LPARAMETERS...`, cu codul din listingul 11.7.

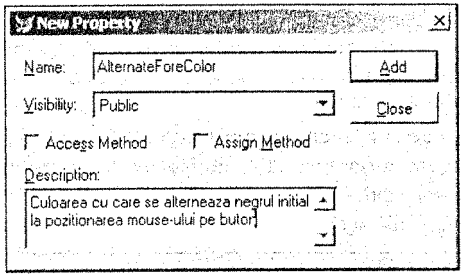


Figura 11.9. Adăugarea unei proprietăți la o clasă existentă

Deoarece proprietatea `AlternateForeColor` are inițial valoarea `.F.`, care nu este compatibilă cu nici o culoare, fiind sursă permanentă de erori, îi vom atribui, încă din clasa-părinte, valoarea numerică a unei culori, obținută prin `=RGB(0,0,255)`. Este culoarea albastră. Precizăm că semnul = din fața funcției este obligatoriu, altfel Visual FoxPro va interpreta funcția `RGB` ca pe un text oarecare, ducând la o nouă eroare.

Listing 11.7. Codul metodei `MouseMove`

```
LPARAMETERS nButton, nShift, nXCoord, nYCoord
WITH THIS
* se memoreaza pozitia relativa a cursorului
* fata de marginile butonului,
* intrucat nXCoord si nY Coord se refera la
* marginile formularului
    horz=nXCoord-.LEFT
    vert=nYCoord-.TOP
    * daca pozitia cursorului se afla intr-o zona delimitata conventional
    IF BETWEEN(horz,.WIDTH*0.2,.WIDTH*0.8) AND ;
            BETWEEN(vert, .HEIGHT*0.2,.HEIGHT*0.8)
        .FORECOLOR=.AlternateForeColor && se inlocuieste culoarea textului
        .FONTBOLD=.T. && se ingroasa fontul, pentru accentuarea efectului vizual
    ELSE
        .FORECOLOR=RGB(0,0,0) && se revine la text de culoare neagra
        .FONTBOLD=.F.
    ENDIF
ENDWITH
```

După terminarea clasei, vom înlocui vechiul buton `Cmdbtnquit1`, de pe formularul `frmChart`, cu un buton din această nouă clasă, al cărui nume implicit va fi `Cmdbtncolorquit1`. Efectul vizual obținut este, credem, destul de plăcut. În plus, la click pe butonul respectiv, formularul va dispărea de pe ecran (se moștenește deci metoda `Click`). Dar dacă, intervenind în clasa `CmdBtnColorQuit`, dorim să atribuim altă funcționalitate evenimentului `Click` (de pildă pentru a utiliza un astfel de control „colorat" și la altceva decât la părăsirea formularului) și scriem acolo doar un comentariu:

```
* evenimentul Click
```

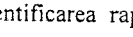
atunci „legătura de rudenie" cu clasa `CmdBtnQuit` se va întrerupe...

În asemenea condiții, pentru a menține funcționalitatea butonului `Cmdbtncolorquit1` de pe formularul `frmChart` nu există altă șansă decât plasarea, în metoda `Click`, a codului din listingul 11.8, în care funcția `MESSAGEBOX()` are rolul explicat mai sus.

Listing 11.8. Apelul explicit al unei metode situate în clasa-părinte

```
#INCLUDE "\aplicatie\constantemsgbox.txt"
IF IDYES=MESSAGEBOX("Abandonati acest grafic?", ;
        MB_YESNO+MB_ICONQUESTION,"Mesaj")
    CmdBtnQuit::CLICK  && se observa apelul unei metode din clasa parinte CmdBtnQuit,
                       && nu din clasa derivata CmdBtnColorQuit
ENDIF
```

Exemplele date aici sunt simple, poate chiar simpliste, dar s-a recurs la simplitate pentru a crește în claritate. Sperăm că vor duce la o mai ușoară înțelegere a modelului de programare obiectuală din Visual FoxPro.

Ați observat probabil că, după crearea celor două clase, bara de instrumente *Form Controls* prezintă două pictograme standard pentru butoane de comandă:  . În astfel de condiții, este greu de ghicit care reprezintă clasa `CmdBtnQuit` și care `CmdBtnColorQuit`, dacă nu se ține un timp cursorul mouse-ului deasupra fiecărui buton. Pentru identificarea rapidă, se recomandă ca fiecărei clase să i se asocieze o pictogramă. Pictograma, cu un aspect sugestiv, trebuie să fie un fișier `.bmp` cu dimensiunile de 15×16 pixeli (dacă e mai mare, Visual FoxPro realizează redimensionarea ei, dar distorsiunile o pot face neinteligibilă). Poate fi desenat și în regim propriu (cu *Paint* sau un program mai sofisticat). Acest fișier trebuie precizat ca valoare a proprietății `Toolbar Icon`, pe care n-o găsim în fereastra obișnuită de proprietăți, ci într-un dialog mai vast (vezi figura 11.10) lansat prin opțiunea de meniu `Class→Class info...` Aceasta este disponibilă doar dacă pe ecran este deschisă fereastra `Class Designer`.

Notă. În afara alegerii din structura arborescentă sub care se prezintă bibliotecile în pagina `Libs` a `Project Manager`, deschiderea rapidă a unei clase în scopul modificării este posibilă și prin scrierea comenzii `MODIFY CLASS ? OF ?`, ce va avea ca efect afișarea unui dialog `Open` precum cel din figura 11.8.

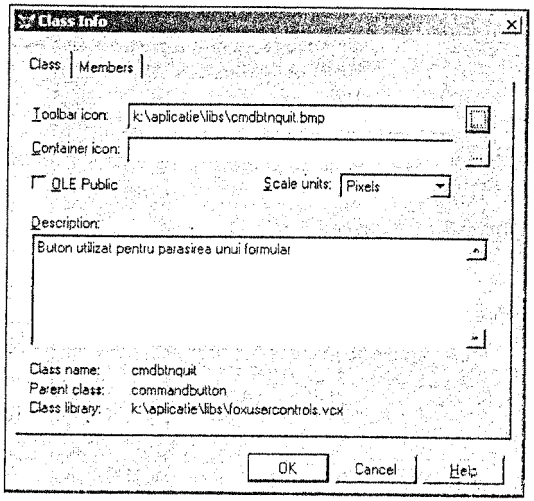


Figura 11.10. Specificarea unei pictograme a unei clase, pentru bara de instrumente

Se poate repeta operația de atribuire a pictogramei și pentru a doua clasă, `CmdBtnColorQuit`. După această manevră, bara de instrumente ar trebui să se prezinte ca în figura 11.11.

Figura 11.11. Bara de instrumente în cazul utilizării pictogramelor (`Toolbar Icon`)

Derivarea clasei Form. Clasa EfectInchidere

În mai toate proiectele Visual FoxPro este nevoie de cel puțin un membru al clasei `Form`, dar sunt proiecte în care este nevoie de mai multe formulare cu un aspect uniform sau cu un comportament anume. Vom continua linia exemplelor simple, oferind indicațiile de creare a unui formular care la închidere să prezinte efectul vizual de „condensare", urmată de dispariție. Ideea este simplă: vom intercepta evenimentul `Unload` și vom diminua progresiv lățimea și înălțimea formularului, cu un procent prestabilit (ne-am oprit la 10%). Diminuarea va avea loc în cadrul unei iterații, până ce lățimea ori înălțimea sunt mai mici de 10% din dimensiunea avută inițial. Pentru exagerarea efectului de animație, am introdus, între două micșorări succesive, un interval exprimat în milisecunde (implicit 100), declarat ca proprietate a clasei, clasă pe care o vom numi `EfectInchidere`.

Listing 11.9. Codul metodei `Unload`

```
SET CURSOR OFF && se ascunde cursorul in timpul executiei instructiunii WAIT
LOCAL i, j, k
WITH THISFORM
    IF .WIDTH>.HEIGHT
        j=.WIDTH
    ELSE
        j=.HEIGHT
    ENDIF
    k=j/10
    FOR i=j TO k STEP -k          && se micsoreaza proportional latimea si inaltimea
                                  && formularului in incremente de 1/10
        .WIDTH=i
        .HEIGHT=i
    ENDFOR
    WAIT WIND "" TIME .INTERVAL/1000      && se asteapta un timp predefinit
    .UNLOAD
ENDWITH
SET CURSOR ON && se reface cursorul
```

Clasa se obține prin derivarea clasei `Form` (vezi lista `Based On...` din figura 11.3), iar memorarea ei are loc tot în biblioteca `foxusercontrols`[57]. Ulterior se adaugă proprietatea `Interval`, iar în procedura evenimentului `Unload` se trece codul din listingul 11.9.

57. La gruparea în biblioteci de clase (`.vcx`) e bine să se țină cont, pe de o parte, de posibilitățile de folosire, într-un singur proiect, a mai multor tipuri de clase „înrudite", iar pe de altă parte, de menținerea unei dimensiuni rezonabile a bibliotecii.

Instanțierea formularului constituie o etapă ceva mai dificilă. Există o cale de a realiza acest lucru în mod asistat. Înainte de alte lămuriri, să spunem că Visual FoxPro utilizează pentru fiecare nou formular (fie el creat cu ajutorul opțiunilor din meniu, fie prin comanda `CREATE FORM`), un șablon (*template*) memorat în biblioteca standard Visual FoxPro. Deși la prima vedere acest lucru nu pare evident, șablonul implicit poate fi modificat, indicând programului un altul, care trebuie să se prezinte ca o clasă `Form` sau derivate din `Form`. Această modificare trebuie realizată înainte de a lansa comanda propriu-zisă de creare a unui nou formular. În acest scop se utilizează opțiunea de meniu `Tools→Options...`, pagina `Forms`. În stânga opțiunii `Form` a cadrului `Template Classes` se găsește o casetă de opțiune pe care o vom bifa (vezi figura 11.12). Se va deschide un dialog `Open` (figura 11.12) în care vom naviga până la găsirea clasei dorite, care trebuie să fie derivată din clasa `Form`. După confirmarea alegerii și închiderea tuturor ferestrelor de dialog, Visual FoxPro va crea orice formular nou pornind de la acest șablon (imaginați-vă utilitatea acestei facilități în caz că ați creat un model de formular sofisticat, cu diferite elemente grafice, culori sau chiar butoane, imagini de fundal și alte controale). Se înțelege de la sine că, formularele deja create rămân neschimbate. Dacă ulterior se dorește revenirea la șablonul implicit pentru formulare, nu trebuie decât debifată căsuța `Form` din dialogul din figura 11.12 (fereastra din plan îndepărtat).

Pentru a ilustra cele spuse, vom crea un formular de probă. În prealabil însă, folosind dialogul arătat în figura 11.12, vom proceda la stabilirea ca șablon a clasei `efectinchidere`, din biblioteca `foxusercontrols`. Apoi, cu comanda `CREATE FORM`, vom purcede la crearea formularului, pe care-l vom și salva cu numele `EfectInchidere.scx`. Nu vom adăuga nici o linie de cod. La execuția formularului, prin click pe butonul de închidere, trebuie să se observe efectul de animație. În continuare se completează formularul, după necesități, cu alte controale. O idee bună este adăugarea unui buton din clasa `CmdBtnQuit`, care va declanșa închiderea (metoda `Release`) și, implicit, animația (reamintim că aceasta este atașată evenimentului `Unload` al formularului).

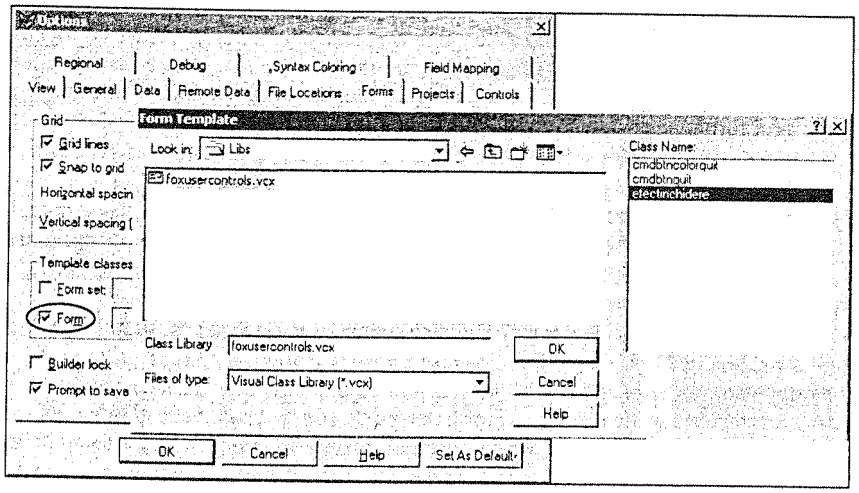


Figura 11.12. Precizarea unui șablon personalizat pentru crearea formularelor

Atenție! Nu trebuie omisă revenirea la șablonul inițial pentru crearea de formulare, prin aceeași opțiune de meniu Tools→Options..., pagina Forms.

Notă. Crearea unei clase derivate din clasa de bază Form poate avea loc și astfel: întâi se creează formularul exact în forma dorită, iar apoi se folosește opțiunea de meniu File→Save As Class... pentru a salva formularul ca și definiție a unei clase. Dialogul Save As Class ne va cere numele clasei, precum și calea și numele unui fișier-bibliotecă .VCX.

Controale de tip Container. Clasele CategPicker și OvalBtn

Existența obiectelor de tip Container permite crearea de clase compuse, prin adăugarea și interconectarea mai multor controale „clasice", fiecare având propriile metode și proprietăți.

Exemplul 1

Vom începe cu o primă clasă, ce folosește controlul TreeView.

Controlul ActiveX TreeView poate afișa o ierarhie, să zicem a unor localități, în care își au sediul clienți ai firmei. Dorim să selectăm doar anumiți clienți, în vederea listării facturilor emise lor. Arborele respectiv fiind organizat pe două niveluri (primul – localități, al doilea – clienți), este posibilă o selecție „granulară": fie toate localitățile și, implicit, toți clienții, fie anumiți clienți (nu toți) din anumite localități (nu toate), fie toți clienții dintr-o localitate etc. Într-un exemplu anterior (formularul frmTree_List din capitolul 10) am avut posibilitatea de a selecta, dintr-o ierarhie clienți-facturi, doar o anumită factură, căreia să i se afișeze detaliile într-o listă alăturată. În ambele cazuri, popularea arborelui se realizează la inițializarea acestuia. Nu s-ar putea crea o clasă care să excludă scrierea de două ori a complicatei proceduri de populare a arborelui cu datele dintr-o tabelă?

O eventuală clasă care să folosească controlul TreeView în scopul precizat ar trebui să permită utilizatorului precizarea sursei de date (tabelă, *view*, interogare SELECT ad-hoc), dar și a ierarhiei în care să se organizeze câmpurile acesteia. Apoi trebuie memorate undeva, la cererea utilizatorului, niște elemente care să identifice unic fiecare ramură care a fost selectată (există un stil de afișare a arborelui ce cuprinde casete de opțiune). În continuare, aceste elemente trebuie să poată fi folosite ca și criterii de filtrare, pentru lansarea unui formular ori raport parametrizat, execuția unei interogări SQL cu parametri în clauza WHERE, deschiderea unui *view* parametrizat etc. Este ideea care a stat la baza clasei prezentate în continuare, numită CategPicker (literal „selector de categorii" – am preferat denumirea englezească pentru conciziunea ei). Astfel, sursa de date, precum și câmpurile necesare ierarhizate și afișate în controlul TreeView vor fi declarate ca proprietăți. Câteva metode adecvate vor asigura popularea arborelui și memorarea identificatorilor ramurilor (nodurilor) alese.

Ce fel de clasă de bază să alegem? De bună seamă, clasa Form ar fi destul de nimerită. Totuși, considerând ideea plasării unui obiect de tip CategPicker pe un formular care conține deja mai multe controale și o cantitate mare de cod, ideea formularului pălește: ne apucăm să refacem (mutăm de pe un formular pe altul, reașezăm, redenumim) mai multe controale deja stabile sau adăugăm un singur control nou? Ultima soluție este mai simplă, dar din păcate controlul TreeView nu are toate proprietățile și metodele pe care le-am dori în acest caz, iar a-l subclasa direct pe el nu poate fi vorba în Visual FoxPro. Cu toate

acestea, este posibil un fel de subclasare, dacă folosim clasa de bază Container, căreia îi putem adăuga orice proprietăți și metode și în care putem include un control de tip TreeView. Dialogul din figura 11.13 arată cum se procedează pentru crearea unei asemenea clase.

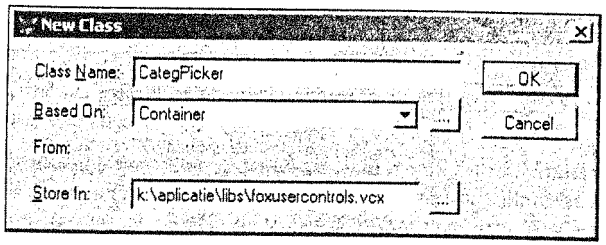


Figura 11.13. Derivarea clasei Container

La încheierea acestui dialog, fereastra Class Designer afișează containerul, foarte asemănător cu un formular, dar fără titlu și fără butoane de control. De fapt, la execuție, controalele de tip container pot avea doar o bordură care să le „trădeze" poziția pe formular.

Vom avea grijă să atașăm la proiect (Tools→Options... Controls) controlul din categoria Microsoft TreeView, iar o instanță a acestui control va fi imediat trasată pe containerul existent (schimbați aspectul barei Form Controls cum se arată în figurile 11.5-11.6). Numele său va fi oleTvw. Vom adăuga și un buton numit cmdMemoreaza, al cărui rol este de a memora într-o tabelă temporară (cursor) cheile ramurilor (nodurilor) care au fost selectate. Respectivele chei, formate din conținutul unor câmpuri din sursa de date, vor putea fi folosite ulterior drept elemente de filtrare.

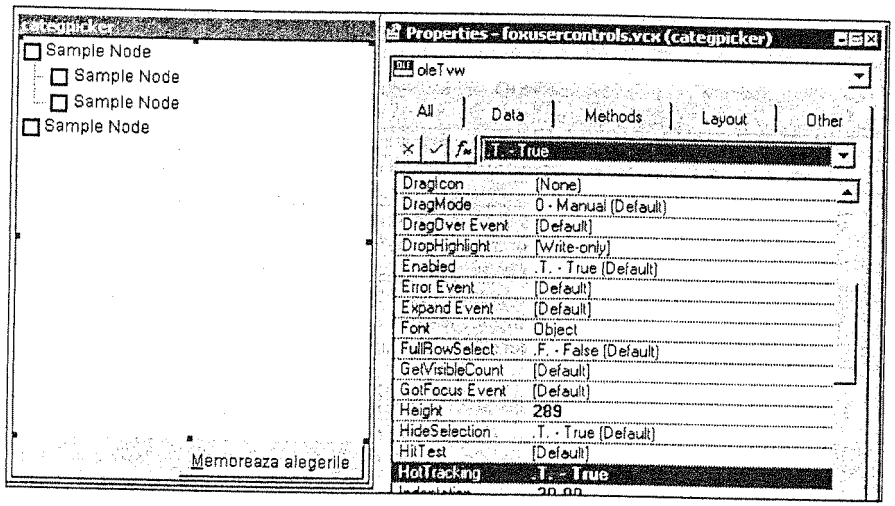


Figura 11.14. Clasa CategPicker în curs de realizare

Tabelul 11.4. Valorile proprietăților notabile ale controlului `oleTvw`

Proprietate	*Valoare*
`Name`	oleTvW
`Checkboxes`	.T.
`HotTracking`	.T.
`Indentation`	20
`LabelEdit`	1 – Manual
`Style`	7 – Treelines, Plus-Minus, Picture and Text

Setarea proprietății `LabelEdit` în acest fel (`Manual`) împiedică modificarea textului de pe o ramură a arborelui, ce se declanșează implicit la execuția a două click-uri succesive (nu dublu-click). Cât despre înălțime și lățime, vom face ca acestea să se ajusteze automat lățimii, respectiv înălțimii containerului (lăsând loc dedesubt pentru butonul `cmdMemoreaza`), folosind evenimentul `Init` (vezi listingul 11.10).

Proprietățile adăugate la nivelul clasei (containerului) sunt precizate în tabelul 11.5. Toate proprietățile sunt de tip `Public`.

Tabelul 11.5. Proprietăți definite de utilizator

Nume	*Descriere*	*Valoare implicită*
`TipSursaDate`	1 – tabelă sau *view*, 2 – interogare SQL	1
`SursaDate`	Un text care să semnifice numele unei tabele sau o interogare `SELECT`. Pentru două niveluri ierarhice, trebuie să avem patru câmpuri în sursa de date. Câmpurile sunt perechi: un cod (o prescurtare, care este ascunsă) + o descriere (care apare pe ramura arborelui). Exemplu: `SELECT l.codpost, loc, codcl, dencl FROM localitati l, clienti c WHERE c.codpost=l.codpost ORDER BY localitate`	<scrieți sursa>
`OrdineCampuri`	Scrieți cifre separate prin virgulă, de exemplu 1,2,3,4, dacă ordinea din tabelă nu este cea potrivită. Întotdeauna codul se precizează înaintea descrierii.	1,2,3,4
`unu`	Primul câmp din sursa de date (ca ordine). Servește pentru memorarea temporară a numelui câmpului-cheie de pe nivelul 1 al arborelui.	.F.
`trei`	Al treilea câmp din sursa de date (ca ordine). Servește pentru memorarea temporară a numelui câmpului-cheie de pe nivelul 2 al arborelui.	.F.

Obiectul `oleTvw`, component al clasei `CategPicker`, a fost declarat de tip `Protected` (vezi dialogul `Class→Edit Property/Method...`). În acest mod nu poate fi accesat explicit în codul instanțelor clasei[58].

58. Reamintim că, în mod implicit, modificarea dispoziției obiectelor dintr-un container se face după activarea opțiunii `Edit` din meniul contextual al containerului. Ei bine, dacă toate obiectele componente sunt de tip `Protected`, opțiunea `Edit` nu mai are efect.

S-au adăugat și două metode, `conversie` și `conversie1`, prevăzute cu sarcini specifice (vezi codul din listingul 11.10).

Listing 11.10. Metodele clasei `CategPicker`

```
                                    * Obiect: CategPicker
* Eveniment: Init
#INCLUDE "\aplicatie\constantemsgbox.txt" && vezi Listing 11.4 si explicatiile aferente
* redimensionarea controlului oleTvw
WITH THIS.oleTvw
    .WIDTH=THIS.WIDTH-2
    .HEIGHT=THIS.HEIGHT-THIS.cmdMemoreaza.HEIGHT-10
    .LEFT=1 && left si top se raporteaza la marginea containerului
    .TOP=1
ENDWITH
* realinierea butonului cmdMemoreaza
WITH THIS.cmdMemoreaza
    .TOP=THIS.oleTvw.TOP+THIS.oleTvw.HEIGHT+5
    .LEFT=THIS.oleTvw.WIDTH-.WIDTH-10
ENDWITH
* cateva verificari ale proprietatilor
IF LEFT(THIS.sursadate,1)="<" && inseamna ca s-a pastrat sursa implicita "<scrieti...>"
    MESSAGEBOX("Sursa de date este incorecta." +CHR(13) + ;
        "Corectati, apoi lansati din nou.", ;
        MB_ICONEXCLAMATION,"Eroare in clasa CategPicker")
    RETURN .F.
ELSE && sursa de date e buna
    IF THIS.tipsursadate=1 AND "SELECT" $ UPPER(THIS.sursadate) ;
            AND "FROM" $ UPPER(THIS.sursadate)
        MESSAGEBOX("Sursa de date este o interogare SELECT," +CHR(13) + ;
            "iar tipul sursei de date este 1. Corectati!", ;
            MB_ICONEXCLAMATION,"Eroare in clasa CategPicker")
        RETURN .F.
    ENDIF
ENDIF
IF !DBUSED('vinzari')
    OPEN DATABASE "\aplicatie\database\vinzari" SHARED
ENDIF
SET EXCLUSIVE OFF
* mai intai, se populeaza controlul de tip TreeView
* pe doua nivele
IF THIS.tipsursadate=1
    IF USED(THIS.sursadate)
        SELECT (THIS.sursadate)
        USE
        USE (THIS.sursadate) IN 0 SHARED ALIAS tabelasursa
    ENDIF
ELSE
* formare comanda SQL
sqlcmd=THIS.sursadate+ " into cursor tabelasursa"
    &sqlcmd          && executie comanda SQL
ENDIF
IF RECCOUNT("tabelasursa")<>0 && exista date de afisat
    * se reordoneaza campurile
    * criteriul e de forma: 1,2,3,4
    * sau 2,1,3,4 sau 1,3,4,2
    * oricum trebuie sa fie numai 4 campuri!
    SELECT tabelasursa
    unu=FIELD(VAL(LEFT(THIS.ordinecampuri, AT(",",THIS.ordinecampuri)-1)))
```

```
    * se memoreaza denumirea si tipul campului-cheie de pe nivelul 1
    THIS.unu=unu+" "+VARTYPE(&unu)+"("+ALLTRIM(STR(FSIZE(unu),3,0))+")"

    doi=FIELD(VAL(SUBSTR(THIS.ordinecampuri, ;
        AT(",",THIS.ordinecampuri)+1, ;
        AT(",",THIS.ordinecampuri,2)-AT(",",THIS.ordinecampuri)-1)))

    trei=FIELD(VAL(SUBSTR(THIS.ordinecampuri, ;
        AT(",",THIS.ordinecampuri,2)+1, ;
        AT(",",THIS.ordinecampuri,3)-AT(",",THIS.ordinecampuri,2)-1)))
    * se memoreaza denumirea si tipul campului-cheie de pe nivelul 2
    THIS.trei=trei+" "+VARTYPE(&trei)+"("+ALLTRIM(STR(FSIZE(trei),3,0))+")"

    patru=FIELD(VAL(SUBSTR(THIS.ordinecampuri, ;
        AT(",",THIS.ordinecampuri,3)+1)))
    * incepe populatea arborelui
    * sa ne inchipuim ca: unu="codpost", doi="loc", trei="codcl", patru="dencl"
    SCAN
        valunu="~"+THIS.conversie(unu, VARTYPE(&unu))
        * s-a concatenat o tilda (~) pentru ca valoarea cheii nodului
        * sa nu inceapa cu cifra, intrucat e incorect
        valdoi=THIS.conversie(doi, VARTYPE(&doi))
        valtrei="~"+THIS.conversie(trei, VARTYPE(&trei))
        * s-a concatenat o tilda (~) pentru ca valoarea cheii nodului
        * sa nu inceapa cu cifra, intrucat e incorect
        valpatru=THIS.conversie(patru, VARTYPE(&patru))
        WITH THIS.oleTvw
            * se adauga o ramura de nivel 1
            IF .nodes.COUNT=0 && initial nu sunt noduri
                * nod-parinte
                .nodes.ADD (,,valunu, valdoi)
                * nod-copil
                .nodes.ADD (valunu,4,valtrei,valpatru)
                anterior=valunu
            ELSE && s-au adaugat noduri
                IF anterior<>valunu && s-a schimbat "parintele"?
                    * un nou nod-parinte
                    .nodes.ADD (,,valunu,valdoi)
                    anterior=valunu
                ENDIF
                * nod-copil
                .nodes.ADD (valunu,4,valtrei,valpatru)
            ENDIF
        ENDWITH
    ENDSCAN
ELSE
    MESSAGEBOX("Nu sunt date de afisat!", MB_ICONINFORMATION)
ENDIF

* Eveniment: Destroy
IF USED("tabelasursa")
    SELECT tabelasursa
    USE
ENDIF

* Metoda Conversie
LPARAMETERS camp, tipcamp
* converteste un camp de un tip precizat intr-un sir de caractere
DO CASE
    CASE tipcamp="C"
```



```
        RETURN ALLTRIM(&camp)
    CASE tipcamp="N"
        RETURN ALLT(STR(&camp))
    CASE tipcamp="D"
        RETURN DTOC(&camp)
    CASE tipcamp="T"
        RETURN TTOC(&camp)
    OTHERWISE
        MESSAGEBOX("Nu stiu sa tratez tipul de camp "+ tipcamp, ;
            MB_ICONEXCLAMATION,"Eroare in clasa CategPicker")
        RETURN "eroare"
ENDCASE

* Metoda Conversie1
LPARAMETERS variabila, tipvariabila
* converteste o variabila de tip sir de caractere
* intr-un tip precizat
DO CASE
    CASE tipvariabila="C"
        RETURN variabila
    CASE tipvariabila="N"
        RETURN val(variabila)
    CASE tipvariabila="D"
        RETURN CTOD(variabila)
    CASE tipvariabila="T"
        RETURN CTOT(variabila)
    OTHERWISE
        MESSAGEBOX("Nu stiu sa tratez tipul de variabila "+ tipvariabila, ;
            MB_ICONEXCLAMATION,"Eroare in clasa CategPicker")
        RETURN "eroare"
ENDCASE

* Obiect: CategPicker.oleTvw
* Eveniment: NodeClick
*** ActiveX Control Event ***
LPARAMETERS node
IF NODE.children<>0 && daca nodul selectat are noduri-copil
    NODE.expanded=!NODE.expanded && inversam starea expandare/comprimare
ENDIF

* Eveniment: NodeCheck
*** ActiveX Control Event ***
LPARAMETERS NODE
LOCAL pc
IF NODE.children<>0
    pc=NODE.CHILD && primul nod copil
    pc.checked=pc.PARENT.checked
ENDIF
FOR i=2 TO NODE.children          && se continua cu selectarea automata
                    && a tuturor nodurilor copil
    pc=pc.NEXT && urmatorul nod copil
    pc.checked=pc.PARENT.checked
ENDFOR
IF NODE.checked
    NODE.expanded=.T.  && expandam/colapsam automat,
                && ca sa se vada nodurile selectate
ENDIF

* Obiect: CategPicker.cmdMemoreaza
* Eveniment: Click
```

```
IF USED("crsSelectie")
    SELECT crsSelectie
    USE
ENDIF
* se formeaza o comada CREATE pentru a defini un cursor cu 2 coloane
* pentru memorarea rezultatelor alegerilor facute in oleTvw
sqlcmd="create cursor crsSelectie("+ THIS.PARENT.unu+","+THIS.PARENT.trei+")"
&sqlcmd   && executie comanda SQL: se creeaza cursorul pentru rezultate
* acum se parcurg ramurile selectate, inserand in cursor
* perechi de forma cheie-nivel1, cheie-nivel2
* reconvertite in tipul de data original
FOR EACH ramura IN THIS.PARENT.oleTvw.nodes
    IF ramura.checked AND VARTYPE(ramura.PARENT)="O" && are parinte
    INSERT INTO crsSelectie VALUES (;
    THIS.PARENT.conversie1(SUBSTR(ramura.PARENT.KEY,2),TYPE(FIELD(1,"crsSelectie"))), ;
    THIS.PARENT.conversie1(SUBSTR(ramura.KEY,2),TYPE(FIELD(2,"crsSelectie"))))
    ENDIF
ENDFOR
```

Exemplificarea modului de folosire a acestui control se bazează pe crearea unui formular experimental numit `TestCategPicker` (figura 11.15).

Prima ipoteză de lucru presupune *afişarea selectivă a facturilor emise unor clienţi din una sau mai multe localităţi*. După cum se observă, singurul obiect existent pe formular este `CategPicker1` (numele implicit atribuit de Visual FoxPro la instanţierea controlului).

Proprietăţile acestui control, `CategPicker1`, se stabilesc astfel:

- `tipsursadate` = 2
- `sursadate = SELECT l.codpost, loc, codcl, dencl FROM localitati l, clienti c WHERE l.codpost=c.codpost ORDER BY loc`
- `ordinecampuri` = 1,2,3,4

Misiunea acestui formular este de a afişa raportul `rep_facturi_clienţi_1`. Este o versiune a raportului `rep_facturi_clienţi`, dezbătut pe larg în capitolul 7 (subcapitolul 7.3). Singura diferenţă o constituie adăugarea următorului cod la sfârşitul evenimentului `Init` al obiectului `Data Environment`:

```
SELECT * FROM craport WHERE codpost IN (SELECT codpost FROM crsSelectie);
    AND codcl IN (SELECT codcl FROM crsSelectie) INTO CURSOR craport
SELECT craport
```

În scopul afişării, evenimentului `Click` al butonului `cmdMemorează` i se adaugă codul prezentat în listingul 11.11. Aceluiaşi buton i se modifică valoarea proprietăţii `Caption` în `\<Afisare raport`.

Notă. La procedura `Click` se ajunge folosind cele două liste din partea superioară a ferestrei de cod.

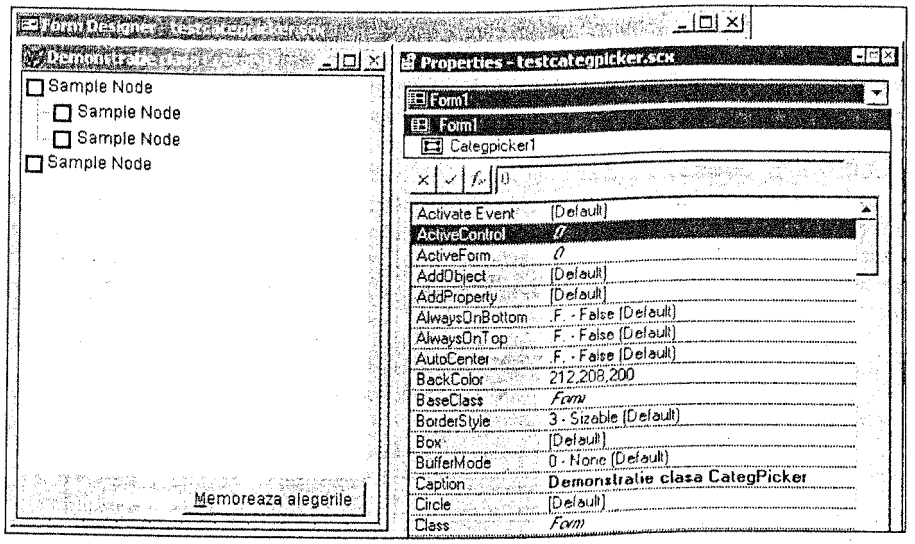


Figura 11.15. Formularul experimental `TestCategPicker`

Listing 11.11. Lansarea raportului

```
* Obiect: thisform.Categpicker1.cmdMemoreaza
* Eveniment: Click
DODEFAULT() && se realizeaza memorarea nodurilor selectate
* se afiseaza raportul in mod previzualizare
REPORT FORM "\aplicatie\reports\rep_facturi_clienti_1" PREVIEW
```

În figura 11.16 se observă raportul în mod previzualizare, ulterior selecţiei câtorva clienţi în arborele din formularul `TestCategPicker`.

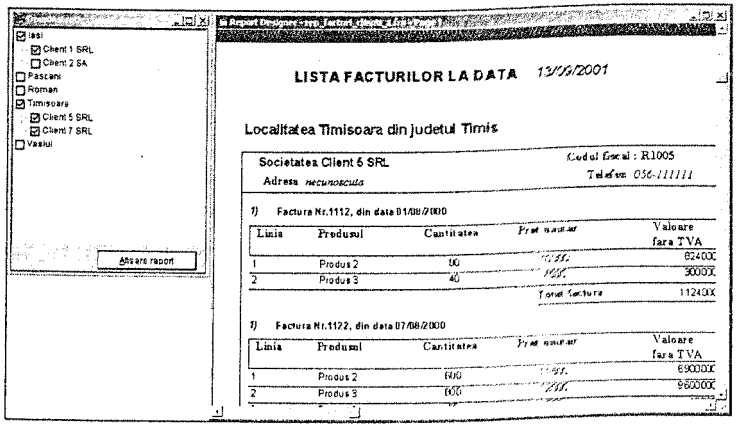


Figura 11.16. Rezultatul execuţiei formularului `TestCategPicker` (varianta 1)

Să recurgem acum la ipoteza potrivit căreia s-ar dori *afişarea numai a anumitor facturi emise unuia sau mai multor clienţi*. Trebuie realizat un nou formular în care, pentru proprietatea `sursadate`, se va stabili textul: `SELECT c.codcl, dencl, nrfact, "Factura nr. "+alltrim(str(nrfact))+" din " +DTOC(datafact) AS denfact FROM clienti c, facturi f WHERE f.codcl=c.codcl ORDER BY c.codcl, nrfact.`

La lansarea formularului, ar trebui să se afişeze fără probleme clienţii pe primul nivel şi facturile pe al doilea (vezi figura 11.17). Dacă apare o eroare, e posibil ca numerele de facturi să nu fi fost unice. Aceasta este de altfel o limitare a clasei: se presupune că perechile de forma (*primul câmp din sursa de date, al treilea câmp din sursa de date*) nu se repetă.

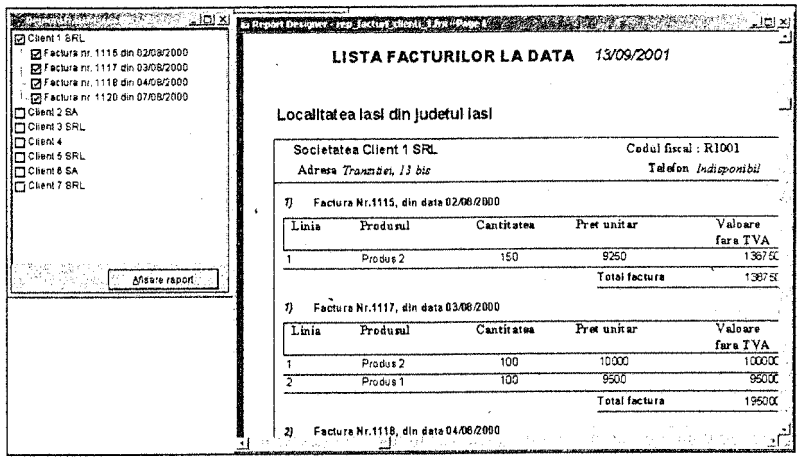


Figura 11.17. Rezultatul execuţiei formularului `TestCategPicker` (varianta 2)

Pentru afişarea corectă a raportului, finalul codului evenimentului `Init` al obiectului `Data Environment` al raportului `rep_facturi_clienţi_1` va arăta astfel:

```
SELECT * FROM craport WHERE nrfact IN (SELECT nrfact FROM crsSelectie);
    AND codcl IN (SELECT codcl FROM crsSelectie) INTO CURSOR craport
SELECT craport
```

Rezultatul poate fi urmărit în figura 11.17, unde au fost selectate toate facturile clientului „Client1 SRL".

Observaţie: am folosit raportul doar în scop demonstrativ, întrucât, în condiţiile filtrării, câmpurile „Total pe client" sau „Total pe localitate" nu vor mai afişa situaţia reală.

Exemplul 2

Vom utiliza un container şi câteva obiecte din clasa `Shape`, în scopul simulării unor butoane de comandă ovale (aşa cum se văd în paginile Web ori la interfaţa grafică a unor sisteme de operare).

Ideea este simplă şi uşor de pus în practică, necesitând cunoştinţe minime de geometrie: se utilizează trei ovale parţial suprapuse: unul cu bordură neagră – pentru conturul butonului, unul gri – pentru umbră şi unul cu bordură punctată, care se va afişa intermitent pentru a simboliza că butonul are focus-ul. Atenţie însă, obiectele de tip `Shape` nu pot primi focus-ul; de aceea s-a ales soluţia „ascunderii" în container a unui obiect `CommandButton` „adevărat", cu dimensiuni care-l fac aproape invizibil (2×2 pixeli). Cât despre textul butonului, el va fi simulat printr-un control `Label` cu proprietăţile `AutoSize=.T.` şi `Alignment=Center`.

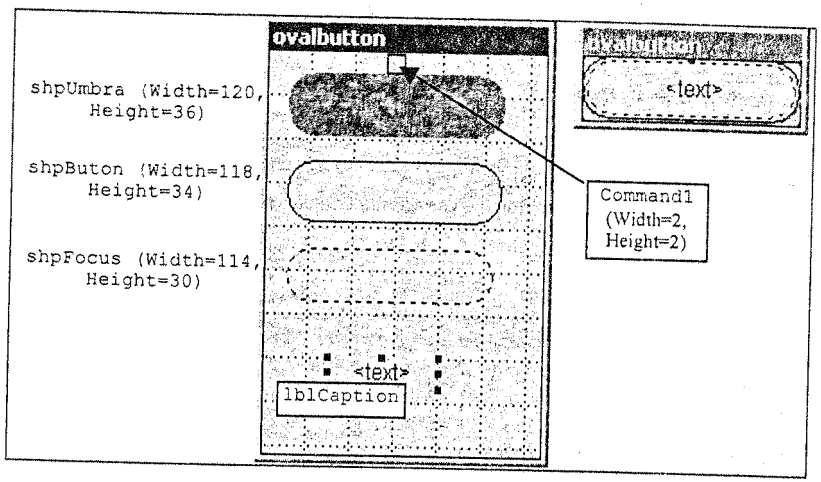


Figura 11.18. Clasa `OvalBtn` în curs de realizare
(stânga – „piesele componente", dreapta – aspectul final)

Iniţial, `shpUmbra` şi `shpButon` au valori egale ale proprietăţilor `Left`, respectiv `Top`, rezultând un efect de umbrire pe laturile din dreapta şi de jos ale butonului. După construcţie, toate controalele incluse în container sunt declarate `Protected`, astfel ca într-o aplicaţie-client, obiectele de tip `OvalBtn` să se comporte monolitic. Tabelul 11.6 prezintă proprietăţile şi metodele adăugate de utilizator.

Tabelul 11.6. Proprietăţi şi metode noi la nivelul clasei `OvalBtn`

Nume	*Descriere*	*Valoare implicită*
`Titlu`	Textul care va fi afişat pe buton	Text
`Curbura_buton`	Un număr de la 1 la 98 (mai mare, înseamnă o curbură mai pronunţată; 0 este dreptunghi; 99 este cerc sau elipsă)	30
`MDn`	Metodă invocată indirect la apăsarea butonului mouse-ului	–
`Mup`	Metodă invocată indirect la eliberarea butonului mouse-ului	–

Metodele `MDn` şi `MUp` sunt gândite ca „destinaţii" ale unor evenimente `MouseDown` şi `MouseUp` provenind de la oricare dintre obiectele componente ale containerului (trebuie ca pseudo-butonul construit aici să reacţioneze la click pe întreaga sa suprafaţă, or aceasta este formată din diferite controale suprapuse). Succesiunea metodelor `MDn` şi `MUp` simulează vizual click-ul. Cât despre evenimentul `Click` ca atare (necesar pentru ca acest buton să

aibă şi un rol util, nu doar decorativ), el este declanşat explicit, la sfârşitul metodei MUp. Vom începe discuţia cu metoda MDn, destinată să provoace un efect vizual prin deplasarea spre dreapta şi în jos, cu câte 1 pixel, a obiectului shpButon, ca şi a textului (obiectul lblCaption), dând iluzia apăsării (listingul 11.12).

Listing 11.12. Codul metodelor MDn, respectiv MUp

```
                    * Obiect: OvalBtn
* Metoda: MDn
LPARAMETERS ctl, nButton, nShift, nXCoord, nYCoord
* ctl este controlul (de tip Shape) "de mutat", transmis ca parametru
IF nButton=1 AND nShift=0
    * se muta "butonul"
    ctl.TOP=ctl.TOP+1
    ctl.LEFT=ctl.LEFT+1
    * se muta textul
    THIS.lblCaption.LEFT=THIS.lblCaption.LEFT+1
    THIS.lblCaption.TOP=THIS.lblCaption.TOP+1
    THIS.command1.SETFOCUS && ca sa se afiseze linia punctata (shpFocus)
ENDIF

* Metoda: MUp
LPARAMETERS ctl, nButton, nShift, nXCoord, nYCoord
* ctl este controlul (de tip Shape) "de mutat", transmis ca parametru
IF nButton=1 AND nShift=0 && butonul din stanga, fara Ctrl Alt Shift
    * se muta la loc "butonul"
    ctl.TOP=ctl.TOP-1
    ctl.LEFT=ctl.LEFT-1
    * se muta la loc  textul
    THIS.lblCaption.LEFT=THIS.lblCaption.LEFT-1
    THIS.lblCaption.TOP=THIS.lblCaption.TOP-1
    THIS.CLICK && se simuleaza click pe container (butonul oval)
ENDIF
```

Parametrii nButton, nShift, nXCoord, nYCoord sunt declaraţi de noi pentru a corespunde unor parametri similari ai evenimentelor MouseDown şi MouseUp (pe scurt, nButton=1 înseamnă apăsarea butonului stâng, dacă nShift=0 înseamnă că nu s-a menţinut apăsată nici una din tastele Shift, Alt sau Ctrl etc. – vezi, în acest sens, documentaţia Visual FoxPro). Parametrul ctl este folosit pentru ca metoda să determine „mişcarea" oricărui control transmis ca parametru (excepţie face controlul lblCaption, care este întotdeauna mutat explicit).

Modul cum aceste metode sunt invocate „dinspre" diferitele controale constituente ale containerului este descris în listingul 11.13.

Listing 11.13. Execuţia metodelor MDn şi MUp

```
                * Obiect: OvalBtn.shpButon
* Eveniment: MouseDown
LPARAMETERS nButton, nShift, nXCoord, nYCoord
THIS.PARENT.mdn(THIS,nButton, nShift, nXCoord, nYCoord)

* Eveniment: MouseUp
LPARAMETERS nButton, nShift, nXCoord, nYCoord
THIS.PARENT.mup(THIS,nButton, nShift, nXCoord, nYCoord)
```



```
                * Obiect: OvalBtn.shpFocus
* Eveniment: MouseDown
LPARAMETERS nButton, nShift, nXCoord, nYCoord
* este necesar in caz ca shpFocus este vizibil,
* deoarece, fiind deasupra, va intercepta click-ul de mouse
THIS.PARENT.mdn(THIS.PARENT.shpButon, nButton, nShift, nXCoord, nYCoord)

* Eveniment: MouseUp
LPARAMETERS nButton, nShift, nXCoord, nYCoord
* este necesar in caz ca shpFocus este vizibil,
* deoarece, fiind deasupra, va intercepta click-ul de mouse
THIS.PARENT.mup(THIS.PARENT.shpButon, nButton, nShift, nXCoord, nYCoord)

                * Obiect: OvalBtn.lblCaption
* Eveniment: MouseDown
LPARAMETERS nButton, nShift, nXCoord, nYCoord
* este necesar in caz ca se executa click chiar pe text,
* deoarece, fiind deasupra, va intercepta click-ul de mouse
THIS.PARENT.mdn(THIS.PARENT.shpButon, nButton, nShift, nXCoord, nYCoord)

* Eveniment: MouseUp
LPARAMETERS nButton, nShift, nXCoord, nYCoord
* este necesar in caz ca se executa click chiar pe text,
* deoarece, fiind deasupra, va intercepta click-ul de mouse
THIS.PARENT.mup(THIS.PARENT.shpButon, nButton, nShift, nXCoord, nYCoord)

                * Obiect: OvalBtn.Command1
* Eveniment: KeyPress
* Nota: evenimentele MouseUp, MouseDown, Click
* nu se trateaza, butonul fiind inaccesibil
* click-ului de mouse
LPARAMETERS nKeyCode, nShiftAltCtrl
* s-a tastat Enter sau Space, deci echivalent cu Click
IF nKeyCode=13 OR nKeyCode= 32
    THIS.PARENT.mdn(THIS.PARENT.shpButon, 1,0,0,0)
    SET CURSOR OFF    && se ascunde cursorul
    WAIT "" TIME 0.1      && se mentine butonul "apasat" 0.1 secunde
                && pentru realizarea iluziei optice
    SET CURSOR ON     && se reface cursorul
    THIS.PARENT.mup(THIS.PARENT.shpButon, 1,0,0,0)
ENDIF
```

Evenimentele GotFocus şi LostFocus ale butonului Command1 au ca efect apariţia, respectiv ascunderea obiectului shpFocus (cel cu chenar punctat, care implicit are proprietatea Visible=.F.) pentru a semnaliza, la execuţia aplicaţiei, că noul control de tip OvalBtn este cel curent.

Listing 11.14. Codul-sursă pentru evenimentele GotFocus şi LostFocus

```
                * Obiect: OvalBtn.Command1
* Eveniment: GotFocus
* se simuleaza un efect de "focus" (linie punctata imprejur)
THIS.PARENT.shpFocus.VISIBLE=.T.

* Eveniment: LostFocus
* se anuleaza efectul de "focus" (linie punctata imprejur)
THIS.PARENT.shpFocus.VISIBLE=.F.
```

Se observă în aceste secvențe de cod adresarea relativă a unor controale, prin sinonimele `THIS` și `PARENT`. Deloc neglijabilă este redimensionarea tuturor controalelor constituente, în caz că se dorește trasarea pe formularul-client a unor butoane mai mici ori mai mari. În acest scop am utilizat metoda `Init` a containerului. Tot aici se citesc și interpretează valorile unor proprietăți precum `Titlu` sau `Curbura_buton`.

Listing 11.15. Codul-sursă pentru metoda `Init`

```
* Obiect: OvalBtn
* Eveniment: Init
* se ascunde bordura containerului (initial e vizibila, fiind utila la redimensionare)
THIS.BORDERWIDTH=0
THIS.lblCaption.CAPTION=THIS.titlu
* alinierea si redimensionarea controalelor
WITH THIS.shpUmbra
    .WIDTH=THIS.WIDTH
    .HEIGHT=THIS.HEIGHT
    .LEFT=0
    .TOP=0
    .CURVATURE=THIS.curbura_buton
ENDWITH
WITH THIS.shpButon
    .WIDTH=THIS.shpUmbra.WIDTH-2
    .HEIGHT=THIS.shpUmbra.HEIGHT-2
    .LEFT=THIS.shpUmbra.LEFT
    .TOP=THIS.shpUmbra.TOP
    .CURVATURE=THIS.curbura_buton
ENDWITH
WITH THIS.shpFocus
    .WIDTH=THIS.shpButon.WIDTH-4
    .HEIGHT=THIS.shpButon.HEIGHT-4
    .LEFT= 2
    .TOP= 2
    .CURVATURE=THIS.curbura_buton
ENDWITH
WITH THIS.lblCaption
    .WIDTH=THIS.shpButon.WIDTH-16
    .LEFT=THIS.shpButon.LEFT+(THIS.shpButon.WIDTH-.WIDTH)/2
    .TOP=THIS.shpButon.TOP+(THIS.shpButon.HEIGHT-.HEIGHT)/2
ENDWITH
WITH THIS.Command1
    .LEFT=THIS.shpButon.LEFT+(THIS.shpButon.WIDTH-.WIDTH)/2
    .TOP=THIS.shpButon.TOP+(THIS.shpButon.HEIGHT-.HEIGHT)/2
ENDWITH
```

Pentru a demonstra utilizarea acestor noi controale, am realizat un formular simplu, legat la tabela `LOCALITATI` și prevăzut cu două butoane „ovale" destinate navigării (vezi figura 11.19).

INFORMATICĂ

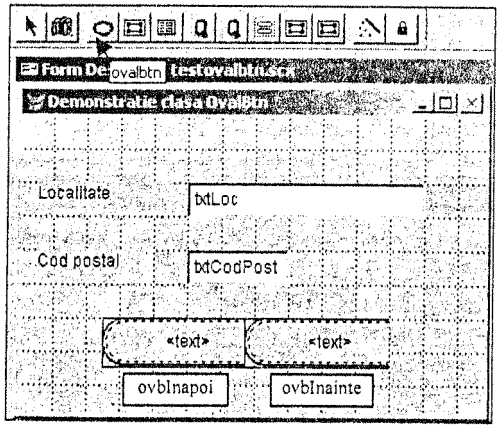


Figura 11.19. Formularul `TestOvalBtn` în faza de proiectare

Tabelul 11.7. Principalele proprietăți ale controalelor de pe formularul `TestOvalBtn`

Proprietate	*Valoare*	*Comentarii*
	`Control: txtLoc`	
`ControlSource`	`Localitati.loc`	Controlul este legat la atributul `loc` din tabela LOCALITATI.
	`Control: txtCodPost`	
`ControlSource`	`Localitati.codpost`	Controlul este legat la atributul `codpost` din tabela LOCALITATI.
	`Control: ovbInapoi`	
`Class`	`OvalBtn`	–
`Caption`	`Ina\<poi`	–
`Curbura_buton`	`30`	–
	`Control: ovbInapoi`	
`Class`	`OvalBtn`	–
`Caption`	`I\<nainte`	–
`Curbura_buton`	`30`	–

Listing 11.16. Codul-sursă al formularului `TestOvalBtn`

```
* Obiect: TestOvalBtn
* Eveniment: Init
* se deschide baza de date
IF !DBUSED("vinzari")
    OPEN DATA "\aplicatie\database\vinzari" SHARED
ENDI
* se deschide tabela LOCALITATI
IF !USED("localitati")
    USE localitati IN 0 SHARED
ENDIF
                                * Obiect: TestOvalBtn.ovbInapoi
* Eveniment: Click
IF !BOF()
    SKIP -1
    THISFORM.REFRESH
ENDIF
```

```
                    * Obiect: TestOvalBtn.ovbInainte
* Eveniment: Click
IF !EOF()
    SKIP
    THISFORM.REFRESH
ENDIF
```

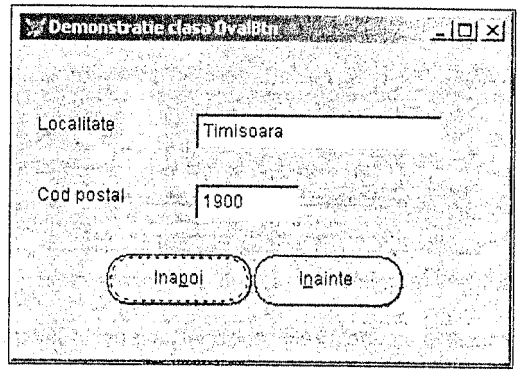


Figura 11.20. Formularul `TestOvalBtn` în faza de execuție

Clasa `Custom` și derivarea ei. Controale „nevizuale"

Clasa `Custom` oferă o foarte mare flexibilitate în crearea unor clase derivate a căror funcționalitate constă în principal în codul-sursă, însărcinat cu calcule complexe, diferite conversii etc., care de regulă se execută „în fundalul" aplicației[59]. Un obiect din clasa `Custom` nu poate conține vreun control. În schimb, clasei `Custom` i se pot adăuga orice proprietăți și metode, pe lângă setul minimal existent. Oricare clasă de bază Visual FoxPro, în afară de `Custom`, poate sta la baza unei clase „nevizuale", cu condiția ca obiectul să fie făcut invizibil (`Vizible=.F.`).

Un exemplu de prelucrare laborioasă, pe care-l vom dezvolta în continuare, este afișarea unui număr (cifră de afaceri, medie a anilor de studii, cotă a Dunării etc.) sub formă de text, ca de exemplu „două sute treizeci" atunci când se introduce 230. Pentru a realiza o astfel de transformare, pot fi imaginați mai mulți algoritmi. Totuși, ideea de bază este urmărirea fiecărei cifre și transcrierea echivalentului său fonetic (zero...nouă). Concomitent, trebuie sesizată poziția (unități – care nu se scrie, zeci, sute, mii...) și adăugat după cifră ordinul respectiv, la genul adecvat („o mie", dar „un milion"). Situația se complică atunci când este vorba de sute de mii, zeci de milioane etc. Mai trebuie ținut cont și de particularitățile lingvistice („șaizeci" nu „șasezeci", de pildă). Algoritmul folosit în continuare nu este dintre cei mai eficienți, dar își atinge scopul. Astfel, numărul, transmis ca parametru, este separat în parte întreagă și parte zecimală, iar fiecare dintre acestea este parcursă, de la prima la ultima cifră, și transcrisă de o procedură specială.

Clasa, pe care o numim `Conversie`, are metodele din tabelul 11.8.

59. „În fundal" este mai curând o figură de stil, întrucât executabilele Visual FoxPro sunt aplicații monofir (*monothreaded*); este vorba de invizibilitatea instanțelor unei asemenea clase.

Tabelul 11.8. Principalele proprietăți ale clasei `Conversie`

Metodă	*Tip acces*	*Descriere*
`Afla_Ordin (Param. Pozitie, Cifra)`	`Protected`	Returnează ordinul cifrei curente („sute", „mii", „milioane" etc.)
`Cifre_In_Litere (Param. Cifra, Pozitie)`	`Protected`	Transcrie cifra curentă în litere
`Transcrie (Param. Portiune, Lungime)`	`Protected`	Formează expresia finală, parcurgând numărul ca șir de cifre, tratând cu cele două metode anterioare fiecare cifră, apoi concatenând cifrele (exprimate în litere) și ordinele corespunzătoare
`Converteste (Param. Numar)`	`Public`	Analizează numărul, îl segmentează în parte întreagă și parte zecimală, le convertește pe fiecare și returnează rezultatele în formă concatenată

Practic, primele trei metode fiind declarate `Private`, numai metoda `Converteste` va fi accesibilă după instanțierea clasei `Conversie`. Fără comentarii suplimentare, redăm mai jos codul metodelor.

Listing 11.17. Codul-sursă al clasei `Conversie`

```
* Metoda: Converteste
LPARAMETERS Numar
LOCAL AreZecimale, pInt, pZec, Intregi, Zecimale
STORE "" TO Intregi, Zecimale
IF Numar-INT(Numar)>0
    AreZecimale=.T.
ENDIF
IF AreZecimale
    pInt = INT(Numar)
    pZec = Numar-INT(Numar)
ELSE
    pInt = Numar
    pZec = 0
ENDIF
pInt=ALLTRIM(STR(pInt))
Intregi=THIS.transcrie(pInt,LEN(pInt))
IF AreZecimale
    FOR i=1 TO 10
        * aici se calculeaza cate zecimale sunt
        * deoarece e nevoie de numarul exact, pentru functia STR()
        IF ROUND(pZec*10^i,i)=INT(ROUND(pZec*10^i,i))
            EXIT
        ENDIF
    NEXT i
    * sunt i+2 cifre in asa-zisa parte zecimala
    pZec=SUBSTR(ALLTRIM(STR(pZec,i+2,i)),3)
    Zecimale=THIS.transcrie(pZec, LEN(pZec))
ENDIF
RETURN Intregi + IIF(!AreZecimale, "", "virgula " + Zecimale)

* Metoda: Transcrie
LPARAMETERS portiune, lungime
LOCAL C2L, ord, litere,cifracrt, cifraurm, lungime
store "" to litere
```

```
FOR i = 1 TO lungime
    cifracrt=SUBSTR(portiune, i, 1)
    cifraurm=SUBSTR(portiune, i + 1, 1)
    ord = THIS.Afla_ordin(lungime - i + 1, VAL(cifracrt))
    C2L = THIS.Cifre_In_Litere(VAL(cifracrt), lungime - i + 1)
    IF LEFT(ord, 3) = "zec"
        DO CASE
        * se rezolva cazurile de forma: unsprezece, paisprezece, saizeci si doi etc...
            CASE cifracrt="1"
                DO CASE
                    CASE cifraurm="1"
                        unitate = "un"
                    CASE cifraurm="4"
                        unitate = "pai"
                    CASE cifraurm="6"
                        unitate = "sai"
                    OTHERWISE
                        unitate = THIS.Cifre_In_Litere(VAL(cifraurm), lungime - i)
                ENDCASE
                IF cifraurm = "0"
                    litere = "zece" + THIS.Afla_ordin(lungime - i, VAL(cifraurm)) + " "
                ELSE
                    litere = litere + unitate + ;
                        "sprezece " + THIS.Afla_ordin(lungime - i, VAL(cifraurm)) + " "
                ENDIF
            OTHERWISE
                DO CASE
                    CASE cifraurm="0" && cand cifra unitatilor este zero
                        litere = litere + C2L + " " + ord
                    OTHERWISE
                        litere = litere + C2L + " " + ord + " si " + ;
                                THIS.Cifre_In_Litere(VAL(cifraurm), lungime - i) + " " +;
                                THIS.Afla_ordin(lungime - i, VAL(cifraurm)) + " "
                ENDCASE
        ENDCASE
        i = i + 1
    ELSE
        litere = litere + C2L + " " + ord + " "
    ENDIF
NEXT i
RETURN litere

* Metoda: Cifre_In_Litere
LPARAMETERS cifra ,pozitie
DO CASE
    CASE cifra=0
        RETURN ""
    CASE cifra=1
        DO CASE
            CASE pozitie=1
                RETURN "unu"
            CASE INLIST(pozitie,3, 4,6,9,11)
                RETURN "una"
            CASE INLIST(pozitie,7, 10, 13)
                RETURN "un"
            OTHERWISE
                RETURN "unu*"      && cazuri exceptionale
        ENDCASE
    CASE cifra=2
        IF pozitie=1
```



```
            RETURN "doi"
        ELSE
            RETURN "doua"
        ENDIF
    CASE cifra=3
        RETURN "trei"
    CASE cifra=4
        RETURN "patru"
    CASE cifra=5
        RETURN "cinci"
    CASE cifra=6
        IF MOD (pozitie,3)=2        && pozitia zecilor (eventual zeci de mii, de milioane etc)
            RETURN "sai"
        ELSE
            RETURN "sase"
        ENDIF
    CASE cifra=7
        RETURN "sapte"
    CASE cifra=8
        RETURN "opt"
    CASE cifra=9
        RETURN "noua"
ENDCASE

* Metoda: Afla_Ordin
LPARAMETERS pozitie, cifra
DO CASE
    CASE MOD(pozitie,3)= 2
        DO CASE
            CASE cifra=0
                RETURN ""
            CASE cifra=1
                RETURN "zece"
            OTHERWISE
                RETURN "zeci"
        ENDCASE
    CASE MOD(pozitie,3)= 0
        DO CASE
            CASE cifra=0
                RETURN ""
            CASE cifra=1
                RETURN "suta"
            OTHERWISE
                RETURN "sute"
        ENDCASE
    CASE MOD(pozitie,3)= 1
        DO CASE
            CASE pozitie=1
                RETURN ""
            CASE pozitie=4
                DO CASE
                    CASE cifra=0
                        RETURN ""
                    CASE cifra=1
                        RETURN "mie"
                    OTHERWISE
                        RETURN "mii"
                ENDCASE
            CASE pozitie=7
                DO CASE
```

```
                    CASE cifra=0
                        RETURN ""
                    CASE cifra=1
                        RETURN "milion"
                    OTHERWISE
                        RETURN "milioane"
                ENDCASE
            CASE pozitie=10
                DO CASE
                    CASE cifra=0
                        RETURN ""
                    CASE cifra=1
                        RETURN "miliard"
                    OTHERWISE
                        RETURN "miliarde"
                ENDCASE
            CASE pozitie=13
                DO CASE
                    CASE cifra=0
                        RETURN ""
                    CASE cifra=1
                        RETURN "biliard"
                    OTHERWISE
                        RETURN "biliarde"
                ENDCASE
        ENDCASE
    ENDCASE
```

Cum se utilizează o asemenea clasă? O aplicație simplă folosește un formular de felul celui prezentat în figura 11.21.

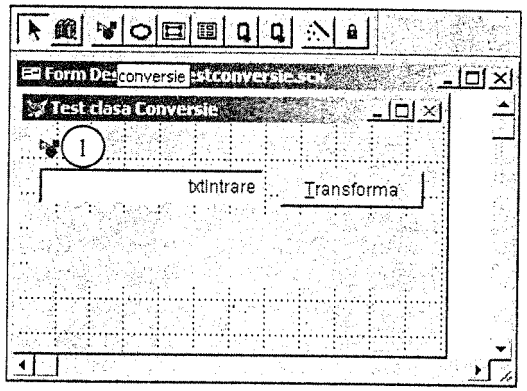


Figura 11.21. Formularul `TestConversie` în faza de proiectare

Se observă în figură că instanța clasei `Conversie` (derivată din clasa `Custom`) afișează un simbol (1) ce permite localizarea sa pe formular. Simbolul corespunde obiectului cu numele implicit `Conversie1` și este vizibil numai la momentul proiectării formularului, nu și la execuție (din acest punct de vedere, seamănă cu controlul `Timer`). Formularul mai conține un control `TextBox` (`txtIntrare`) pentru introducerea unui număr, un control `Label` (`lblIesire`) pentru afișarea numărului în litere. Controlul

`txtIntrare` are ca proprietate `Value=0.000`; controlul `lblIesire` are proprietatea `WrapText=.T.`. Declanșarea conversiei se face cu butonul `cmdTransforma`, a cărui metodă `Click` conține doar secvența:

```
WITH THISFORM
    .lblIesire.CAPTION=""
    .lblIesire.CAPTION=.conversie1.converteste(.txtintrare.VALUE)
ENDWITH
```

Rezultatul execuției formularului se vede în figura 11.22.

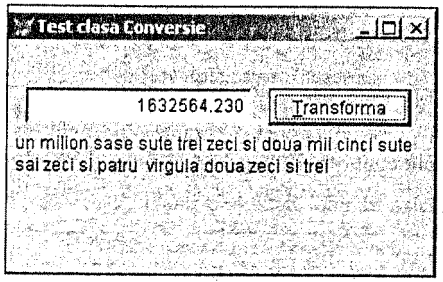


Figura 11.22. Formularul `TestConversie` în faza de execuție

11.2.5. Definirea și instanțierea claselor prin cod

Clasele Visual FoxPro, fie ele vizuale ori nevizuale, pot fi instanțiate și prin cod. Două sunt instrucțiunile-cheie necesare în acest caz:

```
SET CLASSLIB TO <nume-bibliotecă> [ADDITIVE]
```

și

```
<variabilă-de-tip-obiect>=CREATEOBJECT(<nume-clasă>
[,param1[,param2,...]])
```

Comanda `SET CLASSLIB` are rolul de a stabili referința la biblioteca `.VCX` în care este memorată clasa ce se dorește a fi utilizată. Argumentul `<nume-bibliotecă>` este numele fișierului `.VCX`, inclusiv calea, în caz de nevoie. Clauza `ADDITIVE` permite încărcarea bibliotecii în memorie fără a distruge alte referințe la biblioteci deja încărcate. Invers, descărcarea din memorie a unei singure biblioteci `.VCX` are loc prin `RELEASE CLASSLIB <nume-bibliotecă>`, iar a tuturor bibliotecilor se face prin `SET CLASSLIB TO`. În funcția `CREATEOBJECT()`, `param1`, `param2`, ... sunt parametrii transmiși (dacă e cazul) metodei `Init` a clasei care se instanțiază.

Astfel, reconsiderând exemplul anterior, există posibilitatea renunțării la trasarea instanței `Conversie1` direct pe formular, urmând a o crea în metoda `Load` a acestuia.

Listing 11.18. Crearea unei instanțe a clasei `Conversie`

```
* Obiect: TestConversie
* Eveniment: Load
* referinta la biblioteca "foxusercontrols"
SET CLASSLIB TO "\aplicatie\libs\foxusercontrols.vcx" ADDITIVE
* se adauga dinamic formularului proprietatea "Conversie1"
THISFORM.ADDPROPERTY("Conversie1")
* se creeaza obiectul din clasa Conversie
THISFORM.Conversie1=CREATEOBJECT("conversie")
```

Ştergerea referinței la biblioteca `foxusercontrols.vcx` are loc în evenimentul `Destroy` al formularului:

```
* se sterge referinta la biblioteca
RELEASE CLASSLIB "\aplicatie\libs\foxusercontrols.vcx"
```

Cu `CREATEOBJECT()` se poate crea o instanță a unei clase şi într-un program `.PRG`. Oferim cu titlu de exemplu codul din listingurile 11.19. şi 11.20.

Listing 11.19. Utilizarea unei instanțe a clasei `Conversie` într-un program

```
SET CLASSLIB TO "\aplicatie\libs\foxusercontrols.vcx" ADDITIVE
o=CREATEOBJECT("conversie")
?o.converteste(214521.232)
RELEASE o
RELEASE CLASSLIB "\aplicatie\libs\foxusercontrols.vcx"
```

După execuția programului, ecranul va arăta ca în figura 11.23.

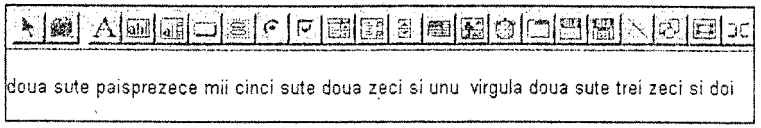


Figura 11.23. Transcrierea în litere a numărului `214521.232`, folosind clasa `Conversie`

Listing 11.20. Utilizarea unei instanțe a clasei intrinsece `Form` într-un program

```
o=CREATEOBJECT("Form")
o.CAPTION="Demonstratie CREATEOBJECT()"
o.HEIGHT=100
o.WIDTH=300
o.SHOW
?"Visual FoxPro"
WAIT "" TIMEOUT 5 && se asteapta 5 secunde
o.RELEASE
```

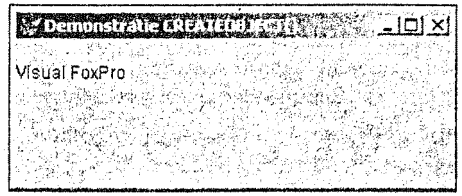


Figura 11.24. Rezultatul execuției programului din listingul 11.20

Funcția `CREATEOBJECT()` se poate folosi şi pentru instanțierea claselor non-Visual FoxPro (vezi listingul 11.21), dar cu sintaxa `CREATEOBJECT(<nume-bibliotecă.nume-clasă>)`, în care numele bibliotecii este cel mai adesea numele unei aplicații care recunoaşte mecanismul *Automation* (numit în trecut `OLE`).

Listing 11.21. Crearea unei instanțe Microsoft Excel şi activarea sa

```
oExcelApp=CREATEOBJECT("Excel.Application") && creare instanta; Excel trebuie sa fie instalat
WITH oExcelApp
    .Workbooks.Add          && creeaza un nou un registru de calcul
    .ActiveWorkbook.Sheets(1).Name="Foaia1"          && redenumeste prima foaie
    .ActiveWorkbook.Sheets(1).Cells(1,1).Value="Hello world!" && afiseaza ceva in casuta A1
    .Visible=.T.            && afiseaza fereastra aplicatiei Excel
ENDWITH
```

Rezultatul este redat în figura 11.25. Excel trebuie închis manual, nefiind precizate instrucțiuni de închidere în listingul 11.21.

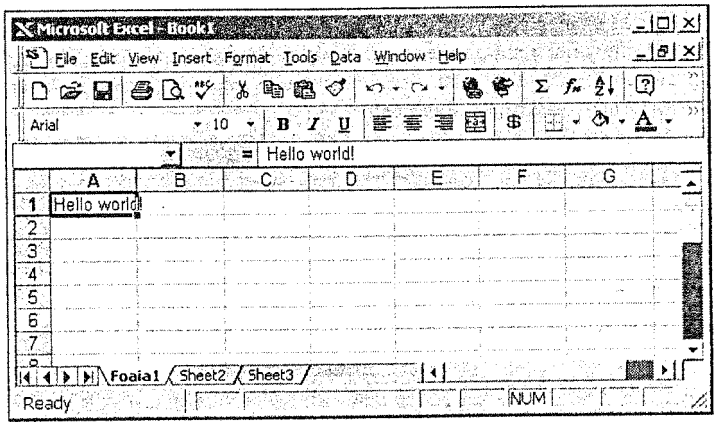


Figura 11.25. Altă utilizare a funcției `CREATEOBJECT()`

Crearea unei clase non-.VCX

Limbajul de programare Visual FoxPro dă şi posibilitatea definirii dinamice a unor clase, prin cod-sursă scris într-un program „obişnuit” `.PRG`. În acest scop, se foloseşte comanda `DEFINE CLASS`, cu sintaxa:

```
DEFINE CLASS <nume-clasa-1> AS <nume-clasa-parinte>
        [OLEPUBLIC]
    [[PROTECTED | HIDDEN <proprietate-1>,
        <proprietate-2> ...]
    [<obiect>.]<proprietate> = <expresie> ...]
    [ADD OBJECT [PROTECTED] <nume-obiect>
        AS <nume-clasa-2> [NOINIT]
    [WITH <lista-proprietati-obiect>]]...
    [[PROTECTED | HIDDEN] FUNCTION | PROCEDURE
        <nume-metoda>[_ACCESS | _ASSIGN]
```

```
        |THIS_ACCESS [NODEFAULT]
    <...codul-sursă al metodei...>
    [ENDFUNC | ENDPROC]]...
ENDDEFINE
```

Câteva remarce (echivalențele cu versiunea „vizuală" sunt date în figura 11.26):

- `OLEPUBLIC` (1) se folosește când clasa respectivă va fi compilată într-o aplicație de tip „Automation Server", adică `.DLL` sau `.EXE`, devenind accesibilă altor aplicații Windows;
- `NOINIT` semnifică faptul că, pentru obiectul constituent, nu se va executa metoda `Init`;
- Numele de forma `proprietate_ACCESS` ori `proprietate_ASSIGN` (2) desemnează o metodă care se execută la citirea, respectiv scrierea valorii proprietății `proprietate`;
- `NODEFAULT` împiedică execuția codului evenimentelor implicite ale obiectelor constituente, la adăugarea lor în definiția clasei;
- `WITH <lista-proprietăți-obiect>` permite specificarea valorilor pentru proprietățile unui control constituent, direct la definirea clasei. Se aplică doar împreună cu `ADD OBJECT` și numai claselor care pot conține obiecte: `Form`, `Container` și eventualelor derivate.

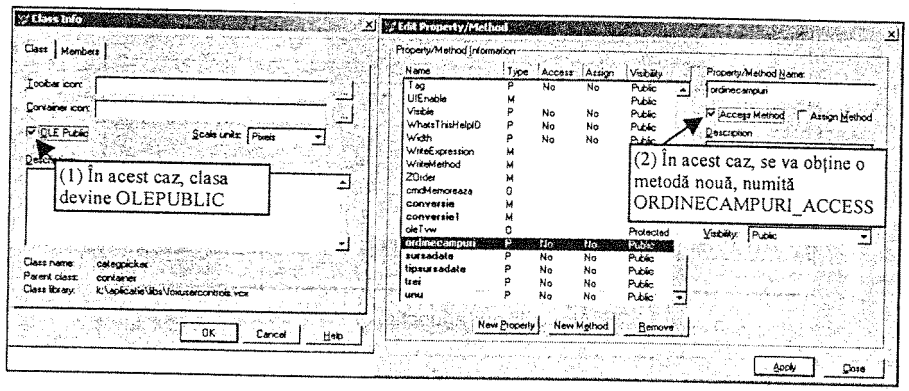


Figura 11.26. Echivalentele „grafice" ale unor opțiuni de definire a claselor

Vom începe cu un exemplu simplu, de clasă nevizuală. Astfel, listingul 11.22 redă un program denumit `clasa_non_vcx.prg`. Din parcurgerea codului se deduce și scopul unei asemenea clase.

Listing 11.22. Definiția clasei `ConvMetric`

```
DEFINE CLASS ConvMetric AS CUSTOM
    NAME="convmetric"
    * conversie centimetri -> inci
    FUNCTION Centimetri_Inci
    LPARAMETERS centimetri
    RETURN centimetri/2.54
    ENDFUNC
    * conversie inci -> centimetri
    FUNCTION Inci_Centimetri
    LPARAMETERS inci
    RETURN inci*2.54
    ENDFUNC
ENDDEFINE
```


Notă. În astfel de programe, orice cod scris după `ENDDEFINE` este ignorat, cu excepția altor definiții de clase. De asemenea, în definițiile de clase nu se pot utiliza structuri de control.

Folosirea clasei (instanțierea sa) implică în prealabil încărcarea în memorie a definiției clasei, prin comanda `SET PROCEDURE` (figura 11.27). La terminarea lucrului, definiția clasei se va elibera din memorie prin comanda `SET PROCEDURE TO`. Cât despre variabila-obiect (`o`, în cazul de față), ea se va distruge la nevoie prin comanda `RELEASE`.

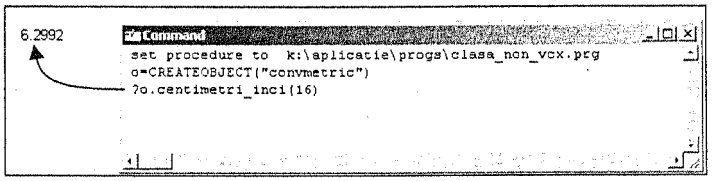


Figura 11.27. Instanțierea unei clase non-`.VCX`

În listingul 11.23 se creează o clasă numită `FormSpec`. Este derivată din clasa `EfectInchidere`, căreia i se adaugă un buton de tip `OvalBtn`, însărcinat cu afișarea unui mesaj. Figura 11.28 arată o instanță a clasei `FormSpec`.

Listing 11.23. O clasă non-`.VCX`, derivată dintr-o clasă `.VCX`

```
SET CLASSLIB TO "\aplicatie\libs\foxusercontrols" ADDITIVE && pentru EfectInchidere si OvalBtn

DEFINE CLASS FormSpec AS efectinchidere
    NAME="formspec"
    AUTOCENTER=.T.
    TITLEBAR=0
    * se adauga un buton oval, cu cateva valori implicite
    * pentru proprietati
    ADD OBJECT ovbGata AS ovalbtn WITH ;
        titlu="I\<esire", ;
        curbura_buton=50, ;
        HEIGHT=30, ;
        WIDTH=90
    PROCEDURE INIT
    * pozitia noului buton
    THISFORM.ovbGata.TOP=THISFORM.HEIGHT-THISFORM.ovbGata.HEIGHT-10
    THISFORM.ovbGata.LEFT=THISFORM.WIDTH-THISFORM.ovbGata.WIDTH-10
ENDPROC
    PROCEDURE ovbGata.CLICK
    RELEASE THISFORM && varianta la thisform.release
ENDPROC
ENDDEFINE
```

11.3. Gestionarea claselor Visual FoxPro

Mediul de dezvoltare Visual FoxPro ne pune la dispoziție mai multe facilități de gestionare a claselor. Ca și clasele, aceste instrumente sunt vizuale (*Project Manager*, *Class Browser*, *Component Gallery*) sau nevizuale (diferite comenzi și funcții).

11.3.1. Instrumente „vizuale" pentru gestionarea claselor

Project Manager ne permite să creăm sau să ștergem biblioteci, să adăugăm, modificăm sau ștergem clase de tip `.VCX`. Este posibilă chiar copierea unei clase dintr-o bibliotecă în alta, prin drag-and-drop (pentru mutare, se va șterge clasa din biblioteca originală).

Class Browser este un alt accesoriu util pentru manipularea bibliotecilor, inspectarea ierarhiilor de clase, vizualizarea și manipularea codului-sursă. Se lansează din meniul `Tools→Class Browser`, iar fereastra sa principală se prezintă ca în figura 11.29.

Cu ajutorul butoanelor din bara de instrumente a *Class Browser* se pot realiza o mulțime de operațiuni asupra bibliotecii curente. O altă bibliotecă poate fi deschisă cu butonul (3) – `View Additional File`, făcând posibile mutări și copieri de clase. Se poate vizualiza și codul, pentru export eventual către un fișier `.PRG`, care să fie folosit altundeva în comanda `SET PROCEDURE TO` (cum s-a arătat anterior).

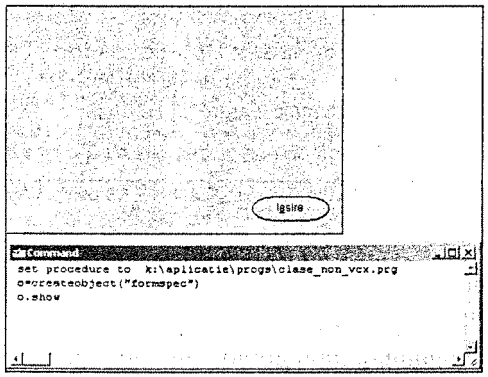


Figura 11.28. O instanță a clasei `FormSpec`

Observație: nu întotdeauna fișierele `.PRG` generate astfel vor funcționa corect (nu funcționează codul claselor vizuale care conțin containere – clasele de bază `Container` sau `PageFrame` – care mai conțin la rândul lor alte controale).

Class Browser dispune și de un meniu contextual, apelabil prin click-dreapta în fereastra din stânga. Acesta permite, printre altele:

- deschiderea unei noi instanțe a *Class Browser* (`New Window`) – deoarece folosirea butonului `Open` (2) închide biblioteca precedentă, putem deschide simultan două biblioteci distincte în două ferestre distincte;
- afișarea membrilor `PROTECTED` sau `HIDDEN` (vor apărea în fereastra din dreapta însoțiți de o steluță);

- lansarea *Class Designer*, pentru clasa pe care s-a efectuat click (`Modify`) – fereastra *Class Designer* se deschide în spatele ferestrei *Class Browser*;
- ștergerea unei clase din bibliotecă (`Remove`).

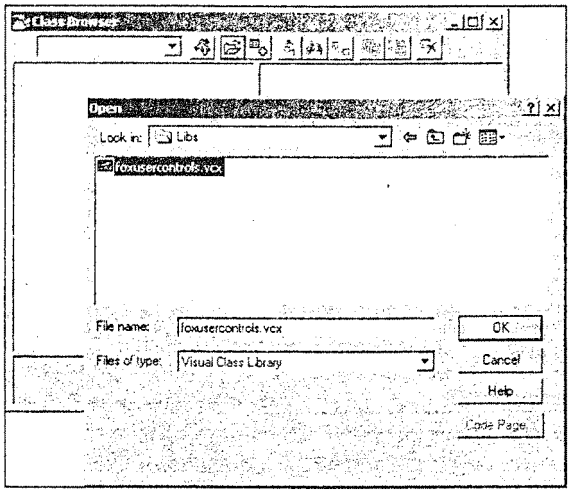


Figura 11.29. Deschiderea unei biblioteci de clase în fereastra `Class Browser`

Observație: operațiile de redenumire, redefinire și ștergere trebuie efectuate cu infinită precauție, mai ales dacă cineva folosește deja respectivele clase într-o aplicație sau aceste clase sunt situate undeva la mijlocul ierarhiei. În primul caz, aplicația-client nu mai recunoaște clasa ștearsă ori redenumită, în al doilea caz, pe lângă clasa „împricinată", sunt pierdute și clasele derivate din ea. Remediu: dacă ați efectuat o modificare neintenționată, închideți pur și simplu biblioteca, răspunzând `No` la întrebarea dacă salvați modificările. Nu mai este nimic de făcut dacă ați acționat între timp butonul (9) – se va lansa comanda `PACK` pe bibliotecă (ea este de fapt o tabelă cu metadate[60]), iar datele șterse vor fi distruse iremediabil. De aceea este bine să aveți întotdeauna o copie a ultimei versiuni funcționale a bibliotecii de clase, într-un director de rezervă.

60. Cei experimentați pot încerca folosirea comenzii `USE <nume-bibliotecă.VCX> IN 0 SHARED`, eventual și a comenzii `BROWSE`, pentru a vedea cum se memorează definiția unei clase. De fapt, în Visual FoxPro cam totul este tabelă: și containerul `.DBC`, și raportul `.FRX`, și formularul `.SCX`.

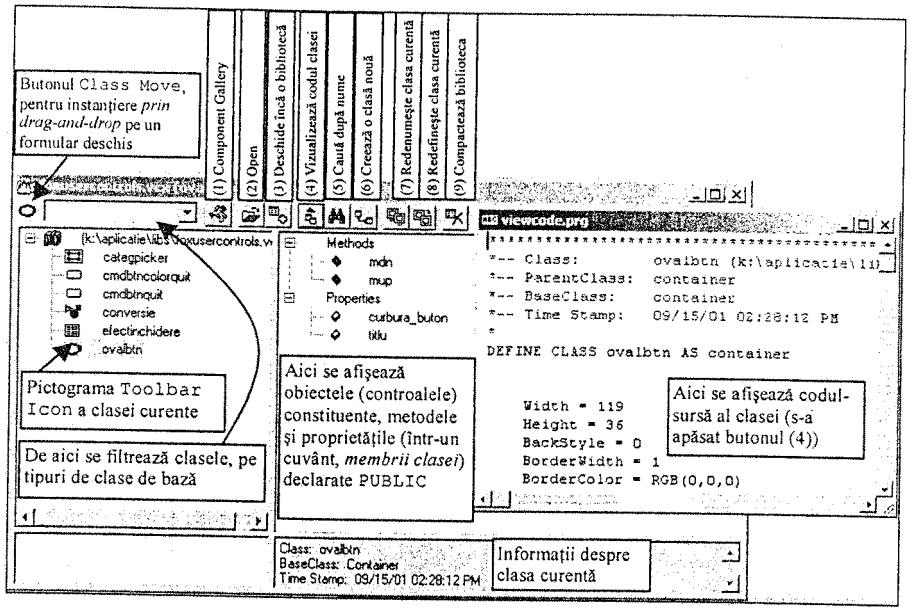


Figura 11.30. Bibliotecă .VCX deschisă în `Class Browser`

Class Browser este un instrument deschis, fiind accesibil (din fereastra de comenzi, de exemplu) prin intermediul variabilei de mediu `_obrowser`. Această variabilă reprezintă de fapt un obiect ce are mai multe metode, cu care poate fi manipulat prin cod. Astfel, dacă sunteți „hobby"-iști ai programării, încercați în fereastra Command următoarea linie de cod:

```
_obrowser.cmdexport.click()
```

Veți constata deschiderea ferestrei cu codul-sursă (evident, *Class Browser* trebuie să fi fost deschis și să se fi selectat o clasă în fereastra lui). De aici, codul poate fi salvat prin comanda `File→Save As...`

Notă. Dacă *Class Browser* este închis, atunci `_obrowser` are valoarea `.NULL.`

O metodă interesantă este `BinToInt()`, ce poate fi folosită drept convertor binar-zecimal ad-hoc. Astfel:

`?_obrowser.bintoint("1010")` va afișa pe ecran valoarea 10.

Pentru detalii cu privire la mulțimea de metode și proprietăți ale obiectului `_obrowser` și utilizarea lor se recomandă consultarea unei documentații detaliate Visual FoxPro.

Component Gallery este un instrument asemănător din punctul de vedere al interfeței (figura 11.31) cu *Class Browser*, dar funcționalitatea sa este diferită. De altfel, se poate trece de la unul dintre aceste două instrumente la celălalt, prin acționarea butonului marcat cu (1) în figurile 11.30 și 11.31. În mod de sine stătător, `Component Gallery` se lansează cu `Tools→Component Gallery`.

Component Gallery funcționează ca un clasor în care sunt grupate și din care pot fi apelate :

- *cataloage de componente* instalate pe sistemul de calcul: lista de componente `ActiveX`, instrumente pentru înregistrarea lor (scrierea informației de identificare în baza de date a sistemului, *Registry*) – catalogul selectat implicit este VFP Main, afișat în figură sub forma folder-ului `Visual FoxPro Catalog`;
- *folder-ul Favorites*, în care programatorul își poate crea shortcut-uri ale componentelor folosite frecvent, pentru a nu pierde timp cu căutarea prin cataloage;
- *catalogul principal Visual FoxPro*, de care ne vom ocupa în continuare, el fiind cel mai vast. Cuprinde:
  - asistenții (*wizards*) pentru: crearea unei baze de date, crearea unei tabele (`Data`), crearea unui formular ori raport cu un anumit aspect predefinit ori chiar a unei aplicații-schelet cu aplicabilitate într-un domeniu (`Templates`) – fiecare asemenea instrument se apelează prin dublu-click;
  - fișiere conținând documentație (`Help`);
  - instrumente de interes general: programele *Calculator*, *Windows Explorer*, *Regedit*, dialogul *FindFile* (`Tools`);
  - clasele fundamentale: (Fox) *Foundation Classes*: un mare număr de clase deja construite, gata de a fi folosite prin simplu drag-and-drop (vezi figura 11.32) din această fereastră spre diferite formulare ori proiecte Visual FoxPro. Clasele sunt categorisite pe destinații. Dacă sunteți interesați de manipularea prin cod a acestor clase, bibliotecile lor sunt stocate pe disc în subdirectorul `\FFC` al directorului-gazdă Visual FoxPro[61]). De exemplu, aveți nevoie de numele bibliotecii dacă doriți să o adăugați barei `Form Controls`, ca să aveți astfel la îndemână controalele din *Foundation Classes*. În catalogul *Foundation Classes* puteți adăuga și clase proprii, dacă deschideți unul din folderele--categorie și alegeți `New Item...` din meniul contextual (click-dreapta).

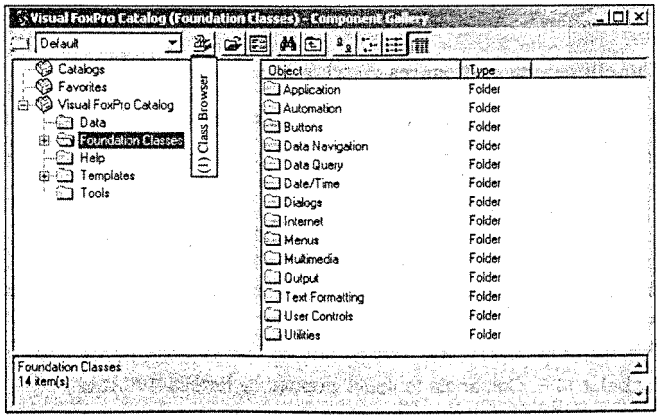


Figura 11.31. Interfața accesoriului `Component Gallery`

61. Puteți afla acest director-gazdă cu funcția `HOME()`.

Notă. Majoritatea acestor clase încep cu caracterul liniuță de subliniere (*underscore*), pentru a arăta că sunt derivate toate din clasele unei biblioteci numite `_base.vcx`. Dacă veți deriva o asemenea clasă pentru nevoi proprii, e bine să stabiliți o convenție clară de denumire (de exemplu, numele clasei să înceapă cu două caractere *underscore*). Dacă doriți să adăugați o proprietate sau metodă generică tuturor claselor care încep cu „_", e bine să o faceți în clasele bibliotecii `_base`; trăsătura se va propaga, în virtutea moștenirii.

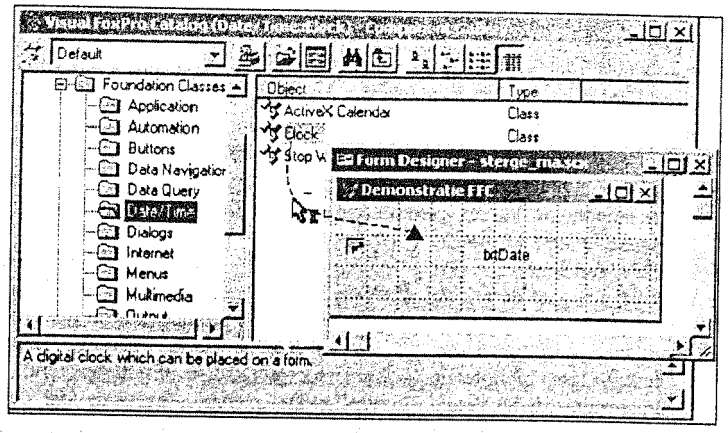


Figura 11.32. Instanțierea clasei `Clock` prin plasare pe un formular

În mod oarecum similar cu *Class Browser*, o referință la *Component Gallery* este memorată în variabila de mediu `_gallery`. Încercați următoarea comandă în fereastra de comenzi: `DO (_GALLERY)`.

11.3.2. Funcții pentru gestionarea claselor

O seamă de funcții Visual FoxPro se adaugă comenzilor `CREATE CLASS`, `MODIFY CLASS`, `REMOVE CLASS`. Aceste funcții sunt utile pentru aflarea unor informații cu privire la bibliotecile și clasele aflate în memorie.

Funcția `AVCXCLASSES(<nume-tablou>,<nume-bibliotecă>)` realizează interogarea fișierului-bibliotecă `.VCX` și plasarea într-un tablou a unor informații utile cu privire la clasă. Tabloul are atâtea linii câte clase are biblioteca; fiecare linie are 11 coloane, ale căror semnificații sunt redate în tabelul 11.9. Rezultatul funcției este chiar numărul de linii, dar acesta interesează mai puțin; important este tabloul însuși. Dacă nu există, el este creat și populat automat.

Tabelul 11.9. Coloanele tabloului populat de funcția `AVCXCLASSES()`

Coloana	*Informații cu privire la clasă*	*Numele câmpului din tabela .VCX*
1	Numele clasei	OBJNAME
2	Numele clasei de bază	BASECLASS
3	Numele clasei-părinte	CLASS

Coloana	*Informații cu privire la clasă*	*Numele câmpului din tabela .VCX*
4	Calea relativă și numele fișierului-bibliotecă	CLASSLOC
5	Calea relativă și numele fișierului-pictogramă al clasei	RESERVED4
6	Calea relativă și numele fișierului-pictogramă pentru o clasă de tip `Project Manager` sau `Class Browser`	RESERVED5
7	Valoarea proprietății `ScaleMode` pentru clasele vizuale: `Pixels` sau `Foxels`	RESERVED6
8	Descrierea clasei (furnizată de utilizator la definirea clasei)	RESERVED7
9	Calea relativă și numele fișierului-header referit prin directiva `#INCLUDE` în codul clasei (dacă există)	RESERVED8
10	Informații despre clasă, definite de utilizator	USER
11	`.T.` dacă clasa a fost declarată `OLEPUBLIC`, `.F.` în caz contrar	RESERVED2

Funcția `ACLASS(<nume-tablou>,<nume-obiect>)` obține informații privind originea obiectului `<nume-obiect>`: clasa din care face parte, clasa-părinte, clasa „părinte a părintelui" etc., până la terminarea ierarhiei de clase. Informațiile sunt scrise într-un vector cu atâtea elemente câte clase sunt în ierarhie.

Funcția `AMEMBERS(<nume-tablou>, <nume-obiect>| <nume-clasă>) [, 1 | 2])` memorează într-un tablou:

- numele tuturor membrilor unei clase sau obiect, dacă ultimul argument este 1 – rezultă un tablou cu două coloane, a doua arătând tipul de membru: proprietate, metodă sau obiect;
- numele obiectelor constituente ale unui obiect de nivel mai înalt (o aplicabilitate imediată este numele tuturor controalelor de pe un formular – vezi listingul 11.24), dacă ultimul argument este 2;
- numele proprietăților unui obiect, dacă se omite ultimul argument.

Listing 11.24. Exemplu de utilizare a funcției `AMEMBERS()`

```
* Demonstratie AMEMBERS() – se lansează un formular oarecare
* Varianta 1
DO FORM "k:\aplicatie\forms\testcategpicker"
oForm=_SCREEN.ACTIVEFORM
=AMEMBERS(aMembriFormular,oForm)
oForm.RELEASE
DISPLAY MEMORY
* Varianta 2
DO FORM "k:\aplicatie\forms\testcategpicker"
oForm=_SCREEN.ACTIVEFORM
=AMEMBERS(aMembriFormular,oForm,1)
oForm.RELEASE
DISPLAY MEMORY
* Varianta 3
DO FORM "k:\aplicatie\forms\testcategpicker"
oForm=_SCREEN.ACTIVEFORM
=AMEMBERS(aMembriFormular,oForm,2)
oForm.RELEASE
DISPLAY MEMORY
RELEASE ALL
```

```
OFORM                                  Pub        O        .NULL
AMEMBRIFORMULAR
                                       Pub        A
      ( 1)                                        C        "ACTIVECONTROL"
      ( 2)                                        C        "ACTIVEFORM"
      ( 3)                                        C        "ALWAYSONBOTTOM"
      ( 4)                                        C        "ALWAYSONTOP"
      ( 5)                  Varianta 1            C        "AUTOCENTER"
      ( 6)                                        C        "BACKCOLOR"
      ( 7)                                        C        "BASECLASS"
      ( 8)                                        C        "BORDERSTYLE"
      ( 9)                                        C        "BUFFERMODE"
      ( 10)                                       C        "CAPTION"
OFORM                                  Pub        O        .NULL
AMEMBRIFORMULAR
                                       Pub        A
  ( 1,  1)                                        C        "ACTIVATE"
  ( 1,  2)                                        C        "Event"
  ( 2,  1)                  Varianta 2            C        "ACTIVECONTROL"
  ( 2,  2)                                        C        "Property"
  ( 3,  1)                                        C        "ACTIVEFORM"
  ( 3,  2)                                        C        "Property"
  ( 4,  1)                                        C        "ADDOBJECT"
  ( 4,  2)                                        C        "Method"
OFORM                                  Pub        O        .NULL
AMEMBRIFORMULAR
                                       Pub        A
      ( 1)                                        C        "CATEGPICKER1"
      ( 2)                  Varianta 3            C        "DATAENVIRONMENT"

  9 variables defined,   41 bytes used
1015 variables available
```

Figura 11.33. Rezultatele celor trei comenzi `DISPLAY MEMORY` din listingul anterior

Funcţia `AINSTANCE(<nume-tablou>,<nume-clasă>)` memorează într-un tablou numele instanţelor unei clase specificate (vezi listingul 11.25 şi figura 11.34). Returnează numărul de instanţe; dacă nu sunt, funcţia returnează rezultatul zero.

Listing 11.25. Exemplu de utilizare a funcţiei `AINSTANCE()`

```
* Exemplu AINSTANCE()
SET CLASSLIB TO "\aplicatie\libs\foxusercontrols" ADDITIVE
oButon1=CREATEOBJECT("ovalbtn")
oButon2=CREATEOBJECT("ovalbtn")
=AINSTANCE(CateInstante, "ovalbtn")
DISPLAY MEMORY
RELEASE CLASSLIB "\aplicatie\libs\foxusercontrols"
RELEASE oButon1, oButon2
```

```
CATEINSTANTE
                                       Pub        A
      ( 1)                                        C        "obuton1"
      ( 2)                                        C        "obuton2"
```

Figura 11.34. Rezultatul comenzii `DISPLAY MEMORY` din listingul anterior

În încheiere, să mai spunem că funcţia `AGETCLASS(<nume-tablou>)` afişează dialogul *Open Class* şi memorează în tablou numele complet al fişierului-bibliotecă şi al clasei pe care le-a ales utilizatorul. Este destul de asemănătoare cu funcţiile `GETFILE()`, `GETPRINTER()`, `APRINTERS()` şi `GETPICT()`. Pentru detalii, se va consulta documentaţia Visual FoxPro.

11.4. Crearea de biblioteci DLL cu Visual FoxPro

Un proiect Visual FoxPro care conţine clase poate fi transformat uşor în *server Automation* (altă denumire este de *server COM*), adică în componentă care să ofere servicii altor aplicaţii.

Se doreşte a se pune clasa `Conversie`, a cărei realizare a fost descrisă mai sus, la dispoziţia oricărei aplicaţii de pe sistem, astfel încât aceasta să poată apela metoda `Converteste` şi deci să afişeze un număr în litere. O bună soluţie este crearea unei biblioteci `DLL` care să includă respectiva clasă. Primul pas constă în crearea unui proiect, la care se ataşează doar biblioteca `FoxUserControls`, ce conţine clasa `Conversie`. Numele proiectului va deveni numele bibliotecii `DLL`, de aceea e bine să alegem un nume sugestiv, să zicem `FoxUserTools.pjx`. Trebuie să reţinem că nu toate clasele unei biblioteci `VCX` vor fi compilate într-o bibliotecă `DLL`, ci numai acelea pentru care s-a specificat tipul `OLE Public`. În acest scop, deschideţi clasa `Conversie`, apelaţi `Class→Class Info...` şi bifaţi căsuţa `OLE Public` (vezi mai sus figura 11.10).

Prin acţionarea butonului `Build` din fereastra *Project Manager*, se deschide dialogul pentru stabilirea opţiunilor de compilare (figura 11.35).

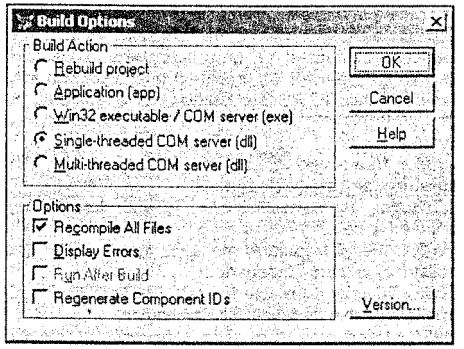


Figura 11.35. Stabilirea configuraţiei de compilare

Alegeţi opţiunea *Single-threaded COM server (dll)* şi apăsaţi `OK`. După ce precizaţi un nume al fişierului `.DLL` destinaţie, va începe generarea bibliotecii. Cu această ocazie, se introduc şi informaţiile de identificare a bibliotecii în baza de date a sistemului, numită *Registry*.

O bibliotecă `DLL` (*Dynamic Link Library*) conţine secvenţe de cod cu multiple destinaţii (proceduri, funcţii sau... definiţii de clase), care pot fi încărcate în spaţiul de memorie al unui program executabil apelant, fie acesta conceput cu Visual FoxPro sau cu orice alt limbaj. De aceea, ele se mai numesc şi componente *in-process*. Prin contrast, programele cu extensia `.EXE` sunt componente *out-of-process*, întrucât se execută într-un spaţiu de memorie separat de cel al programului apelant (s-a demonstrat mai sus crearea şi manipularea unei instanţe a programului Excel, în care programul `EXCEL.EXE` joacă rolul de server *out-of-process*).

Încărcarea unei funcții („pe stil vechi") dintr-o bibliotecă DLL se face prin comanda DECLARE <numefunctie> IN <nume-bibliotecă> <listă-de-parametri>. Ulterior, funcția de bibliotecă se apelează ca o funcție obișnuită Visual FoxPro. Pentru a „descongestiona" memoria după folosirea bibliotecii, trebuie dată comanda CLEAR DLLS (vezi listingul 11.26).

Listing 11.26. Apelul unor funcții Windows dintr-o bibliotecă DLL

```
*********************************************************************************
* Exemplu de apel biblioteca DLL
*********************************************************************************
* pentru formatele functiilor GetComputerName si respectiv GetUserName
* consultati documentatia Windows SDK (MSDN) pe disc sau pe Internet
* http://msdn.microsoft.com/library/
*********************************************************************************
LOCAL UserName, ComputerName
LOCAL Buffer_ComputerName, Lungime_Buffer_ComputerName
LOCAL Buffer_UserName, Lungime_Buffer_UserName
Buffer_ComputerName=SPACE(20)
Lungime_Buffer_ComputerName=20
Buffer_UserName=SPACE(20)
Lungime_Buffer_UserName=20
* Declar functia GetComputerName cu alias-ul "W" (in caz de Unicode, diacritice s.a.)
* Cere transmiterea de pointeri la ambele argumente (transmitere prin adresa sau referinta)
DECLARE GetComputerName IN  kernel32 AS GetComputerNameW STRING @ , LONG @
* Declar functia GetUserName cu alias-ul "W" (in caz de Unicode, diacritice s.a.)
* Cere transmiterea de pointeri la ambele argumente (transmitere prin adresa sau referinta)
DECLARE GetUserName IN advapi32 AS GetUserNameW STRING @ , LONG @
*********************************************************************************
* invoc functia GetUserName
=GetUserNameW(@ Buffer_UserName, @ Lungime_Buffer_UserName)
* sirul se termina cu nullstring, pe care-l indepartam
UserName=LEFT(ALLT(Buffer_UserName),LEN(ALLT(Buffer_UserName))-1)
* invoc functia GetComputerName
=GetComputerNameW(@ Buffer_ComputerName, @ Lungime_Buffer_ComputerName)
* sirul se termina cu nullstring, pe care-l indepartam
ComputerName=LEFT(ALLT(Buffer_ComputerName),LEN(ALLT(Buffer_ComputerName))-1)
CLEAR DLLS && nu mai am nevoie de functiile de biblioteca
MESSAGEBOX("Sunteti " + UserName +", la statia " +ComputerName, 64,"Informatie")
```

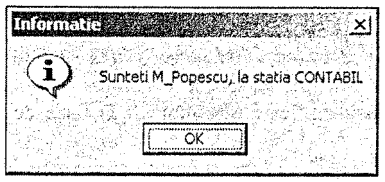


Figura 11.36. Rezultatul execuției programului din listingul 11.26

Astfel se procedează pentru funcții. O clasă compilată într-o bibliotecă DLL nu mai are nevoie de comanda DECLARE, ci poate fi invocată direct prin:

<variabilă>=CREATEOBJECT(<nume-bibliotecă>.<nume-clasă>)

Iată un exemplu în figura 11.37.

Totuși, la ce ne folosește o astfel de mostră de program: din moment ce clasa rezidă într-o bibliotecă .VCX, nu se putea oare crea o instanță a sa în manieră tipică Visual FoxPro,

cu comanda SET CLASSLIB urmată de apelul funcției CREATEOBJECT()? Ba da, dar puterea bibliotecilor DLL constă în oferirea de componente „gata făcute" nu exclusiv aplicației „creatoare", aici Visual FoxPro, ci oricărei aplicații, în orice limbaj care suportă mecanismul *Automation*. Iar exemplul din figura 11.38 arată exact acest lucru, folosind limbajul Visual Basic (funcția CREATEOBJECT() are în Visual Basic exact rolul pe care-l are în Visual FoxPro).

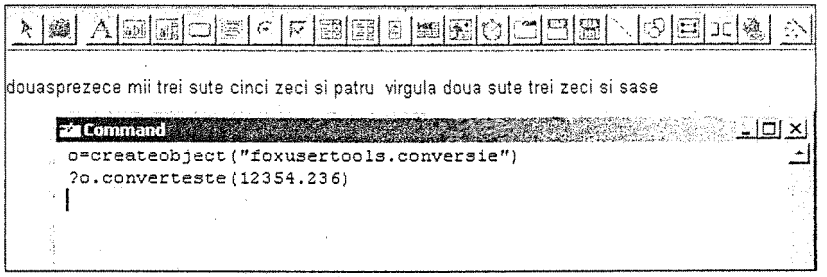


Figura 11.37. Apelul metodei Converteste a clasei Conversie din biblioteca FoxUserTools.dll, după crearea unei instanțe a acesteia

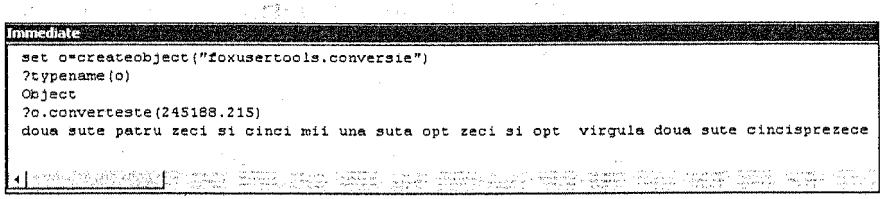


Figura 11.38. Apelul din fereastra Immediate (Visual Basic) a metodei Converteste a clasei Conversie din biblioteca FoxUserTools.dll

Capitolul 12

Meniuri în VFP

Din perspectiva contactului cu utilizatorul, punctul de plecare sau poarta către funcționalitatea practică a unei aplicații, prin obiecte cum sunt formularele sau rapoartele, presupune existența unui element de interfață care să concentreze și să structureze căile de acces în „universul" sistemului. În etapa de dezvoltare, instrumentul `Project Manager` asigură navigarea și accesul la structurile interne ce formează aplicația. În etapa de exploatare, cel mai des folosite instrumente de acces și navigare către sau prin structura funcțională a aplicației sunt meniurile însoțite eventual de bare de instrumente personalizate. Meniul integrează de fapt toate celelalte elemente de interfață și definește pentru utilizator trăsăturile sau caracteristicile generale ale aplicației.

12.1. Crearea meniurilor folosind generatorul de meniuri al Visual FoxPro

Pentru construirea unui *meniu* nu ne vom „adânci" în sintaxe de comenzi specifice, ci vom merge pe o cale mult mai simplă, propusă de mediul VFP prin instrumentul `Menu Designer` (asistentul de proiectare a meniurilor). Este adevărat, la „începuturile" FoxPro-ului, singura modalitate de a defini opțiunile unui meniu, modul de structurare a acestora și acțiunile ce urmau să aibă loc la selectarea uneia dintre ele era scrierea unui program în care se invocau comenzi de genul `DEFINE PAD, DEFINE POPOUP, DEFINE BAR, ON SELECTION BAR` și altele de acest gen. Mai târziu a fost integrat în FoxPro un strămoș al generatorului de meniuri de astăzi și anume `Menu Builder`, a cărui sarcină era crearea programului-sursă pentru definirea meniului și compilarea lui. În Visual FoxPro, instrumentul `Menu Designer` are aceeași sarcină ca și predecesorul lui, rezultatul final concretizându-se în principal într-un fișier cu extensia `.MPR` (programul--sursă) și un fișier `.MNX` (forma compilată).

Procesul de creare a unui meniu pentru întreaga aplicației sau doar pentru o componentă a ei poate fi structurat în cinci etape:

1. Planificarea și proiectarea meniului, adică luarea deciziei asupra structurii generale – sau, altfel spus, „care sunt submeniurile și opțiunile acestora ?".
2. Crearea efectivă a meniurilor și submeniurilor, adică declararea, folosind `Menu Designer`, a titlurilor opțiunilor din bara principală, a opțiunilor corespunzătoare submeniurilor ș.a.m.d.
3. Specificarea acțiunilor ce vor fi declanșate prin selectarea opțiunilor meniului, care ar putea însemna afișarea formularelor, rapoartelor sau lansarea în execuție a unor proceduri ce realizează sarcini specifice. În plus, se poate include o secvență de cod pentru inițializarea mediului înainte de activarea propriu-zisă a meniului sau a unei secvențe de cod pentru restabilirea – „curățarea" – mediului după părăsirea meniului (dezactivarea/ștergerea lui din memorie).
4. Generarea programului (a secvenței de comenzi `DEFINE PAD, DEFINE POPOUP, DEFINE BAR, ON SELECTION BAR` corespunzătoare componentei meniului din cadrul proiectului aplicației).
5. Rularea și testarea programului generat anterior.

12.1.1. Planificarea meniului aplicației

Prin urmare, înainte de a începe propriu-zis construirea meniului aplicației, este recomandabil, la fel ca în cazul formularelor sau rapoartelor, schițarea submeniurilor principale și opțiunilor acestora, din cel puțin două motive: (1) structurarea opțiunilor din meniu în așa fel încât să reflecte o anumită logică funcțională, ușor de înțeles din partea utilizatorilor care vor exploata efectiv aplicația; și (2) luarea în considerare a tuturor funcțiilor aplicației, pentru a se evita omiterea accidentală a unei componente, compilate și deci parte integrată în sistem, dar imposibil de accesat.

Fără a avea pretenția că ar fi cea mai bună structură funcțională (o mare importanță aici o are partea de analiză și proiectare care ar trebui să evidențieze principalele funcții sau „cazuri de utilizare" ale sistemului), propunem următorul format pentru meniul principal al aplicației dezvoltate în cursul capitolelor anterioare:

- meniul-bară va fi compus din următoarele submeniuri:
  - *Nomenclatoare* – care va permite accesul către formularele ce actualizează datele din tabelele de referință cum sunt `Persoane, Clienti` sau `Produse`;
  - *Operațiuni* – care va permite accesul la formularele responsabile de principalele tranzacții economice deservite prin aplicație, adică facturări și încasări;
  - *Liste/Informații* – care va cuprinde o serie de opțiuni prin care utilizatorul va avea acces la informațiile necesare din sistem. Prin urmare, vor fi invocate în primul rând rapoartele predefinite (parametrizate sau nu) sau alte obiecte grafice cum ar fi, de exemplu, diagramele tip chart;
  - *Administrare/Configurare* – care va fi alcătuit dintr-o serie de opțiuni ce vor avea legătură cu întreținerea bazei de date (crearea copiilor de siguranță/arhivarea și refacerea/restaurarea bazei de date și indecșilor – problemă acută în VFP) și configurarea mediului software/hardware în care rezidă aplicația (rețea, imprimantă etc.);
  - *Asistență* – care va constitui o punte către sistemul de *help* al aplicației;
  - *Ieșire* – care va desemna căile prin care se poate părăsi aplicația.

Schema de bază a meniului principal ar putea arăta ca în figura 12.1.

Nomenclatoare	*Operațiuni*	*Liste/Inform.*	*Admin./Config.*	*Asistență*	*Ieșire*
Persoane	Facturari	Lista clienti	Admin. BD	Conținut și index	
Produse	Incasari	Lista facturi	Config. Retea		
Clienti		Raport vânzări	Config. Imprimanta		
		Alte informații			

Figura 12.1. Schema inițială a meniului principal

12.1.2. Crearea/definirea meniului principal şi submeniurilor corespunzătoare

Lansarea generatorului de meniuri al VFP se poate face în două moduri. Pe de o parte, din `Project Manager` se poate selecta din tab-ul `Other` nodul `Menus` (în cazul în care se doreşte modificarea unui meniu deja creat, se expandează acest nod şi se selectează numele acestuia) şi apoi se „apasă" butonul `New` (sau `Modify`, pentru un meniu existent). Pe de altă parte, se poate executa comanda `CREATE MENU` din fereastra de comenzi a VFP.

Atenţie! Din fereastra ce se lansează după apelarea butonului `New` *se va merge pe varianta* `Menu`.

Lucrul cu instrumentul `Menu Designer` este relativ simplu, principala dificultate (oarecum) fiind dispunerea viitoarelor opţiuni din bara principală într-o coloană verticală şi nu orizontală (vezi figura 12.2).

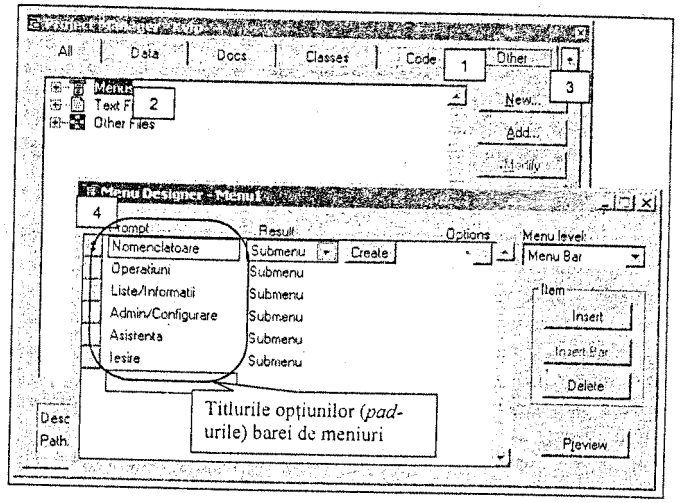


Figura 12.2. Lansarea instrumentului `Menu Designer`

Definirea efectivă a meniului începe mai întâi cu bara pricipală (`Nomenclatoare`, `Opţiuni`, `Liste...`), care va înlocui în momentul execuţiei meniul sistem (`File`, `Edit`, `View...`). Astfel, în coloana `Prompt` a `Menu Designer`-ului se vor consemna titlurile care vor desemna opţiunile ce vor forma meniul principal, iar în coloana `Result` se va specifica ce acţiune va avea loc ca urmare a selecţiei fiecărei opţiuni. Lista derulantă din fiecare linie a coloanei `Result` poate lua una dintre următoarele valori:

- `Submenu` – permite crearea unui submeniu care va fi activat în urma selecţiei respectivei opţiuni (în coloana din dreapta va apărea un buton `Create`, prin „apăsarea" căruia se va deschide calea către specificarea unui submeniu);
- `Command` – permite specificarea în coloana din dreapta a unei comenzi VFP obişnuite, care va fi executată în urma selectării opţiunii curente;
- `Procedure` – permite specificarea codului-sursă pentru un program VFP care va fi lansat în execuţie ca urmare a selectării respectivei opţiuni (în coloana din dreapta va apărea un buton `Create`, acţionarea acestuia determinând lansarea unei ferestre de editare cod-sursă);
- `Pad Name` – numele intern al unei opţiuni.

Revenind în contextul exemplului nostru, în coloana `Prompt` se vor specifica titlurile: `Nomenclatoare`, `Operatiuni`, `Liste/Informatii`, `Admin/Configurare`, `Asistenta`, `Iesire`. Pentru toate aceste titluri se va specifica drept rezultat `Submenu`, cu excepţia opţiunii `Iesire`, pentru care se va specifica `Command`, iar pe coloana a treia se va scrie deocamdată instrucţiunea `SET SYSMENU TO DEFAULT`, din raţiuni de simplificare a procesului de testare-depanare. Această instrucţiune determină revenirea la meniul principal al sistemului Visual FoxPro. Ulterior, în faza finală, se poate înlocui această comandă cu instrucţiunea `QUIT`, care determină părăsirea efectivă şi imediată a mediului VFP.

Trecerea pe nivelul următor în definirea unei opţiuni din meniul-bară se face apăsând butonul `CREATE` sau `EDIT` (pentru revenirea într-un submeniu creat anterior). În lista combinată situată în colţul dreapta-sus a `Menu Designer`-ului şi etichetată `Menu Level` va apărea numele submeniului ale cărui opţiuni sunt create/editate la un moment dat. Această listă permite revenirea pe nivelul (nivelurile) anterioare. Prin urmare, navigarea printre submeniurile aplicaţiei se poate realiza în `Menu Designer` prin butoanele `CREATE/EDIT` (în adâncime) şi prin lista `Menu Level` (înapoi) – vezi figura 12.2.

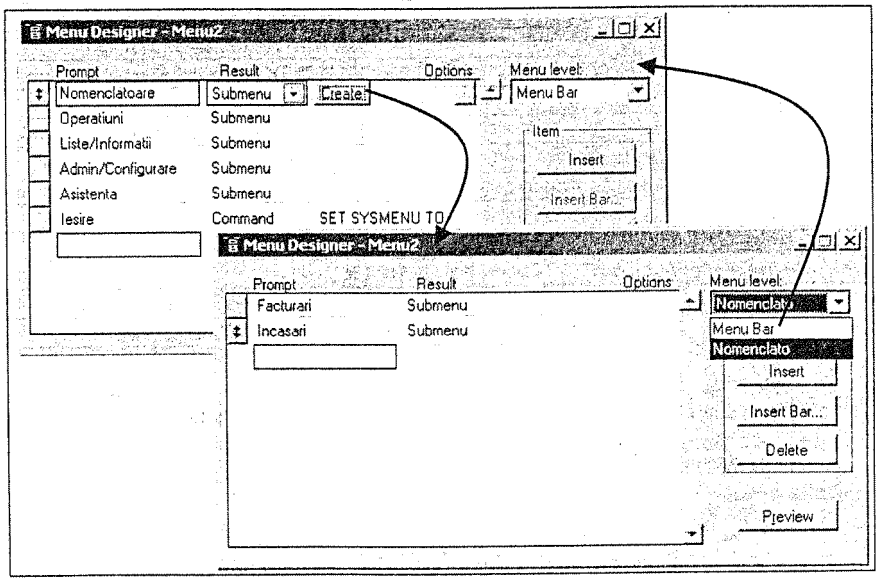


Figura 12.3. Calea de navigare de pe un nivel superior pe un nivel inferior (meniu – submeniu) şi invers

Ţinând cont de aceste precizări, se pot implementa cu uşurinţă în `Menu Designer` specificaţiile din tabelul de 12.1.

Tabelul 12.1. Specificații pentru meniul aplicației

Titlu opțiune sau submeniu	Rezultat	
Meniul-bară	Submenu	
Nomenclatoare	Submenu	
Persoane	Command	**do form frmPersoane**
Produse	Command	**do form frmProduse**
Clienti	Command	**do form frmClienti**
Operatiuni	Submenu	
Facturari	Command	**do form FrmLiniiFact**
Incasari	Command	**WAIT WINDOW "Under construction"**
Liste/Informatii	Submenu	
Lista clientilor	Command	**report form rep_clienti preview**
Lista facturilor	Command	**report form rep_facturi_clienti preview**
Lista vinzarilor pe clienti	Command	**report form rap_dinamic preview**
Informatii din facturi	Command	**do form frmtree_list**
Administrare/Configurare	Submenu	
Administrare BD	Submenu	
Refacere indecsi	Command	**do refacere_indecsi**
Verificare indecsi	Command	**do restrictii_referentiale_macro**
Copie de siguranta BD	Command	**WAIT WINDOW "Under construction"**
Restaurare BD	Command	**WAIT WINDOW "Under construction"**
Configurare retea	Command	**WAIT WINDOW "Under construction"**
Configurare imprimanta	Submenu	
Imprimanta	Bar #	**_mfi_sysprint**
Pagina	Bar #	**_mfi_pgset**
Asistenta	Submenu	
Continut și index	Command	**HELP**
Despre	Command	**WAIT WINDOW "Under construction"**
Iesire	Command	**SET SYSMENU TO DEFAULT**

Se poate trage cu ușurință concluzia că specificațiile de mai sus sunt caracteristice unei etape intermediare în dezvoltarea aplicației; opțiunile cu rezultatul `WAIT WINDOW "Under Construction"` semnifică faptul că „încă se mai lucrează" și pot fi incluse eventual într-un prototip de evaluare-testare din partea utilizatorilor, nicidecum în faza finală de livrare.

12.1.3. Căi suplimentare de acces de la tastatură la opțiunile din meniu și separarea opțiunilor în cadrul submeniurilor

În faza de proiectare-planificare, pentru o împărțire judicioasă a opțiunilor, se poate cere împărțirea acestora în grupuri chiar în interiorul submeniului din care fac parte, fără a se recurge la noi subniveluri (nu se recomandă crearea a mai mult de trei-patru niveluri

consecutive). De asemenea, pentru apelarea rapidă a opțiunilor celor mai des folosite se recomandă să li se asocieze combinații de taste cu rol de chei de acces.

Accesul opțiunilor din meniu se poate face în două moduri: (a) la nivelul submeniului și (b) *shortcut*-uri de la tastură, fără să fie necesară navigarea până la nivelul submeniului din care face parte opțiunea vizată.

Asocierea unei chei de acces a unui submeniu sau opțiuni se realizează introducând simbolurile „\<" înaintea literei care va juca acest rol. Spre exemplificare, vezi figura 12.4.

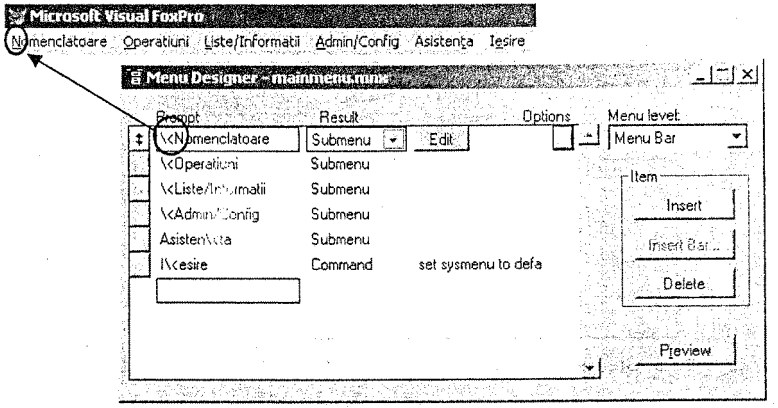


Figura 12.4. Transpunerea unei chei de acces în bara de meniu la „runtime"

Definirea unei combinații de taste care să acceseze direct o opțiune din meniu, fără să fie necesară o navigare explicită, se realizează astfel (vezi figura 12.5):

1. Se selectează numele opțiunii respective (coloana `Prompt`).
2. Se acționează butonul din coloana `Options` pentru activarea dialogului `Prompt Options`.
3. Se selectează căsuța `Key Label` și se apasă combinația de taste dorită pentru activarea respectivei opțiuni.
4. În căsuța `Key Text` se specifică textul care va însoți în meniu numele opțiunii respective.

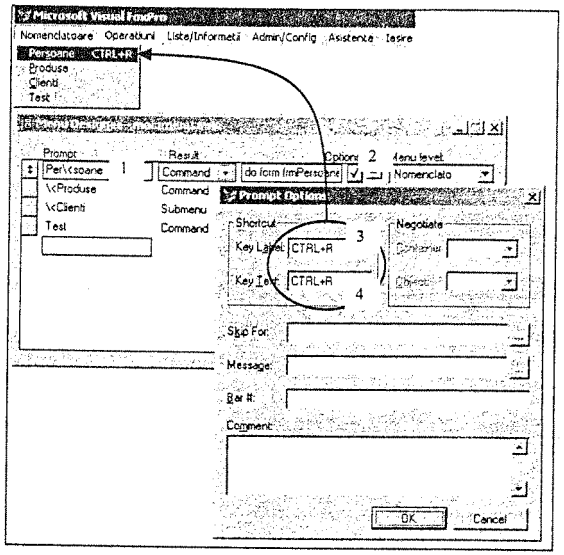


Figura 12.5. Crearea unui *shortcut* de la tastatură şi transpunerea „runtime"

Împărţirea în grupuri distincte a opţiunilor care formează un submeniu este posibilă introducând în fereastra de editare a `Menu Designer`-ului a unei opţiuni suplimentare pentru care este precizat ca titlu (în coloana `Prompt`) şirul „\-", după cum se exemplifică în figura 12.6.

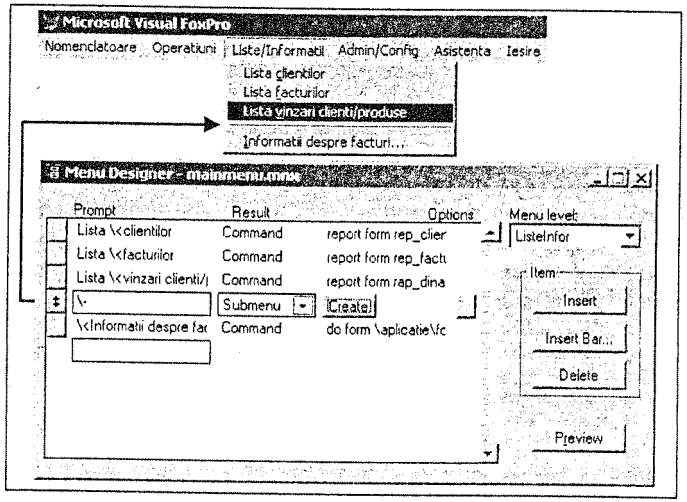


Figura 12.6. Separarea opţiunilor unui submeniu

12.1.4. Includerea în meniul aplicaţiei a unor submeniuri `system`

În anumite situaţii, pentru definirea unor sarcini specifice ale aplicaţiei este necesară „împrumutarea" unor funcţionalităţi proprii sistemului Visual FoxPro. În exemplul nostru, pentru definirea parametrilor de configurare ai imprimantei sau pentru definirea unor parametri referitori la aşezarea în pagină este utilă invocarea în meniul aplicaţiei a opţiunilor corespunzătoare din meniul `system` al mediului Visual FoxPro.

În acest sens, dacă revenim în tabelul 12.1, vom observa că opţiunile

`Administrare\Configurare→Configurare Imprimantă→Imprimanta` şi

`Administrare\Configurare→Configurare Imprimantă→Pagina`

au ca rezultat `Bar#` cu `_mfi_sysprint`, respectiv `_mfi_pgset`.

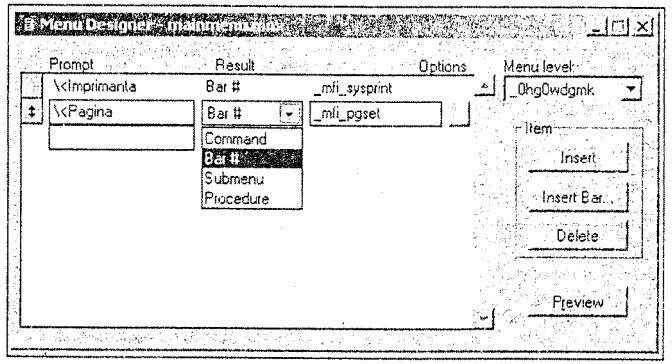


Figura 12.7. Invocarea unor meniuri `system` ca opţiuni ale meniului aplicaţiei

Cele două variabile nu reprezintă altceva decât numele intern al opţiunilor `File→Print` şi `File→Page Setup` ale meniului standard VFP (`_msysmenu`).

În *MSDN Library – Menu menu, Visual FoxPro system (internal)* pot fi găsite numele interne ale tuturor opţiunilor şi submeniurilor care formează meniul `system` VFP. De asemenea, aceste nume interne pot fi afişate şi folosind funcţia `SYS(2013)`.

12.1.5. Dezactivarea opţiunilor dintr-un meniu

La execuţie, există situaţii în care este necesară dezactivarea unor opţiuni sau submeniuri, sau activarea acestora numai în funcţie de anumite condiţii. În general, acest lucru este util atunci când se construieşte o politică de securitate pentru aplicaţie, astfel încât activarea opţiunilor care formează meniul să fie făcută în funcţie de numele utilizatorilor care lucrează la un moment dat sau în funcţie de privilegiile/drepturile asociate acestora.

În exemplul nostru, ar fi necesară o astfel de funcţionalitate pentru a se evita afişarea mesajelor `Under construction` pe care le-am asociat iniţial opţiunilor aflate încă „în lucru".

Această funcţionalitate poate fi realizată întorcându-ne în fereastra `Prompt Option` apelând-o pentru opţiunile vizate să aibă un asemenea comportament. În această fereastră se poate remarca existenţa rubricii `Skip For`. Dacă expresia specificată în acestă rubrică

este evaluată ca fiind `.T.`, respectiva opțiune va fi dezactivată; dacă însă este evaluată ca fiind `.F.`, opțiunea va fi activată (vezi figura 12.8).

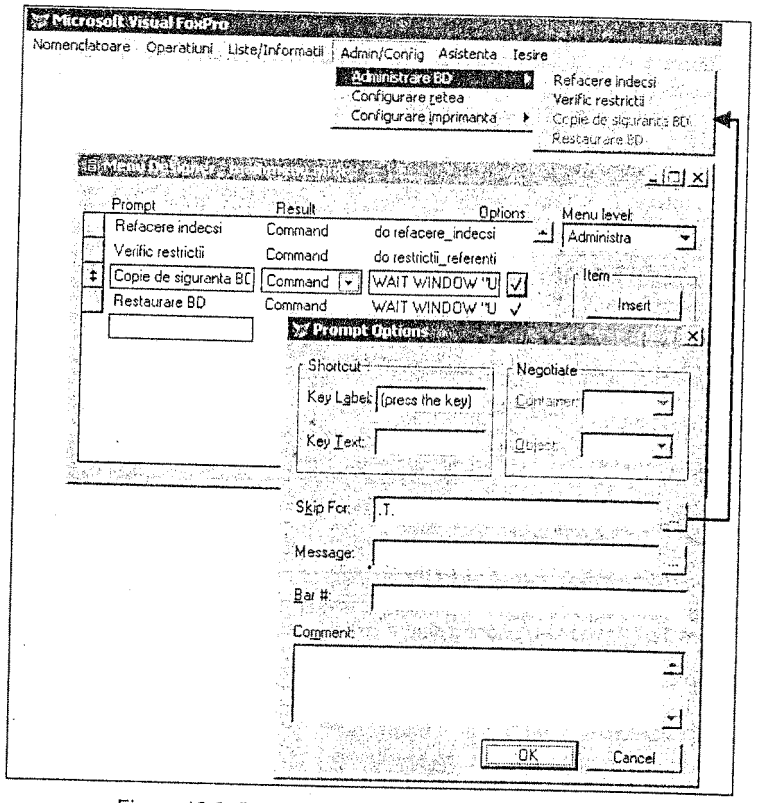


Figura 12.8. Dezactivarea unor opțiuni sau submeniuri

12.1.6. Asocierea directă a unor module de program – cod-sursă – la nivelul meniurilor

Scrierea unor module cod-sursă se poate face, atunci când se consideră necesar, și la nivelul meniurilor, astfel încât aceste fragmente de cod vor fi incluse în programul generat ca urmare a specificațiilor precizate prin intermediul `Menu Designer`-ului.

În general, există trei modalități prin care se poate realiza acest lucru:

- asocierea (scrierea) unor proceduri la nivelul opțiunilor individuale (precizând pentru coloana `Result` alternativa `Procedure`) – cod care va fi apelat la „runtime" ca urmare a selecției din meniu a repectivelor opțiuni;
- asocierea (scrierea) unor module de program la nivelul submeniurilor care vor forma meniul principal;

- asocierea (scrierea) unor module de program la nivelul general al meniului – *setup* și *cleanup code*.

De exemplu, pentru definirea unei funcționalități simple prin care să se interogheze specific baza de date (cum ar fi cererea de afișare a situației încasărilor pe facturi), putem declara o nouă opțiune `Liste/Informații → Situația încasărilor` căreia să îi asignăm codul-sursă specific direct în meniu, după cum se ilustrează în figura 12.9.

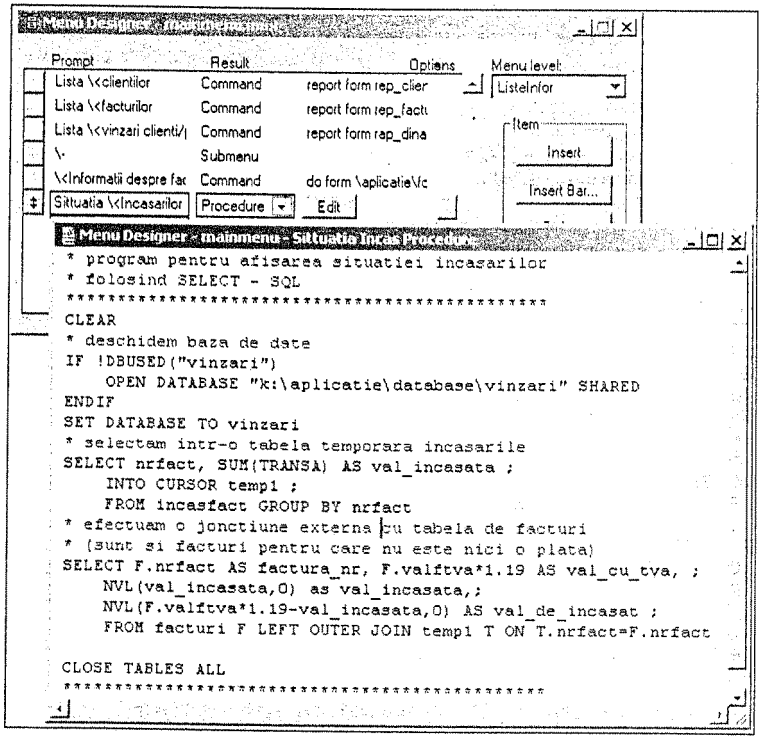


Figura 12.9. Scrierea unei proceduri asignate unei opțiuni din meniu

Există și situații în care asignarea unei proceduri la nivelul unui submeniu se poate dovedi utilă pentru aplicație. De exemplu, opțiunile meniului `Admin/Config→ Administrare BD` pot fi apelate numai dacă baza de date este deschisă în mod exclusiv sau dacă, eventual, poate fi deschisă astfel. Prin urmare, apelarea fiecăreia dintre ele ar trebui să verifice sau să deschidă exclusiv baza de date. Codul care ar trebui să efectueze această sarcină poate fi asignat direct la nivelul submeniului `Administrare BD`.

În acest scop, scriem mai întâi această procedură ca un program de sine stătător cu numele `deschid_exclusiv` (vezi listingul 12.1).

Listing 12.1. Codul-sursă al procedurii `deschid_exclusiv`

```
** Procedura DESCHID_EXCLUSIV ----------------------------------
*** Functiile apelate prin optiunile acestui submeniu
*** pot fi efectuate numai daca BD este deschis execlusiv
PUBLIC vEroare
LOCAL vdbname, vExclusive
ON ERROR DO proc_err
vEroare = 0
vdbname = 'VINZARI.DBC'
*** Verificam daca BD este deschisa
IF !dbused(vdbname)
  IF MESSAGEBOX('Baza de date nu este deschisa. Continuati ?',;
     4+32, 'Confirmare?') = 6
    OPEN DATABASE &vdbname EXCLUSIVE
*** In caz de eroare la deschiderea BD renunt
   IF !EMPTY(vEroare)
     ON ERROR
     RETURN
   ENDIF
  ELSE
    ON ERROR
    RETURN
  ENDIF
ELSE
*** Verificam daca BD este deschisa exclusiv
  IF !ISEXCLUSIVE('vinzari', 2)
    IF MESSAGEBOX('Baza de date nu este deschisa exclusiv. Continuati ?',;
       4+32, 'Confirmare?') = 6
      CLOSE DATABASES
      OPEN DATABASE vinzari EXCLUSIVE
*** In caz de eroare la deschiderea BD renunt
      IF !EMPTY(vEroare)
        ON ERROR
        RETURN
      ENDIF
    ELSE
      ON ERROR
      RETURN
    ENDIF
  ENDIF
ENDIF
*** Functie de optiunea apelata a meniului
*** execut procedura asociata
DO CASE
CASE BAR() = 1
      DO refacere_indecsi
CASE BAR() = 2
  DO restrictii_referentiale_macro
CASE BAR()= 3
  WAIT WINDOW "Under construction"
CASE BAR() = 4
  WAIT WINDOW "Under construction"
ENDCASE
*-------------------------------------------------------------
PROCEDURE proc_err
IF ERROR() = 1705
  vEroare = 1705
  MESSAGEBOX('Baza de date nu poate fi deschisa exclusiv !')
```



```
** Dezactiv inclusiv meniul
  SET SKIP OF POPUP administra .t.
ENDIF
ON ERROR
ENDPROC
```

După care deschidem instrumentul `Menu Designer` navigând până la nivelul meniului vizat. În fine, din meniul standard VFP se apelează opțiunea `View→Menu Options`, iar în fereastra deschisă, la rubrica `Procedure`, se precizează comanda `DO deschid_exclusiv` (vezi figura 12.10).

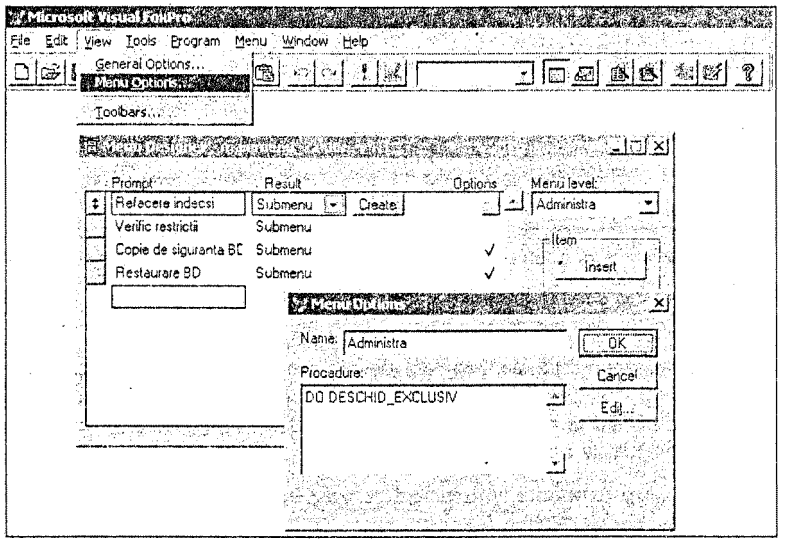


Figura 12.10. Asignarea unei proceduri la nivel de submeniu

Trebuie remarcate două lucruri:

- tuturor opțiunilor meniului `Administrare BD` le-am anulat procedurile inițiale (sunt declarate ca submeniuri);
- funcționalitatea acestor opțiuni a fost „mutată" în codul programului `deschid_exclusiv`, care este executat la selectarea oricărei opțiuni a submeniului. Determinarea opțiunii selectate efectiv se realizează folosind funcția `BAR()` (vezi listingul 12.1).

Am procedat astfel deoarece comenzile (din programul de definire a meniului) prin care sunt apelate funcțiile opțiunilor, de exemplu

```
ON BAR 1 OF administra do refacere_indecsi
```

au întâietate la execuție asupra comenzii prin care este asociată procedura `DESCHID_EXCLUSIV` la nivelul submeniului:

```
ON SELECTION POPUP administra do DESCHID_EXCLUSIV.
```

Codul procedurii `Deschid_exclusiv` exemplificate mai sus poate fi scris și direct la nivelul submeniului cu titlul `Administrare`, fiind astfel inclus în întregime în codul meniului principal (nefiind invocat în acest caz printr-o comandă specială `DO`

<procedură>), după cum este exemplificat în figura 12.11. Acest lucru se realizează alegând butonul Edit din dialogul Menu Options.

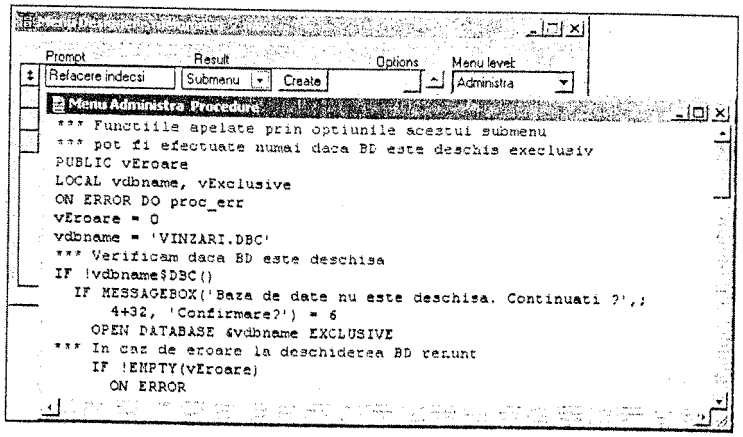


Figura 12.11. Asocierea codului-sursă invocat de un submeniu direct la nivelul acestuia

În fine, asocierea modulelor cod-sursă la nivelul general al meniului principal se realizează prin intermediul opțiunilor Setup Code și Cleanup Code.

Codul Setup reprezintă un fragment de program care se va executa înainte de formarea efectivă a meniului ca urmare a execuției programului rezultat din compilarea fișierului .mpr în care este definit meniul (acesta conține comenzile de definire generate ca urmare a specificațiilor din Menu Designer).

Codul Cleanup reprezintă un fragment de program care se execută după încheierea execuției meniului definit prin fișierul .mpr.

Când se poate dovedi utilă folosirea unor asemenea programe? Situația cea mai evidentă ar fi atunci când meniul reprezintă „catalizatorul" întregii aplicații, execuția acesteia începând și încheindu-se cu formarea și închiderea meniului principal. Prin urmare, meniul principal reprezintă în această situație și programul principal al aplicației. În aceste circumstanțe s-ar dovedi utilă stabilirea configurației mediului în funcție de specificul aplicației înainte de formarea meniului (deci prin intermediul codului setup) și restabilirea configurației inițiale după eliberarea (închiderea) meniului (prin intermediul codului cleanup). De exemplu, în codul setup am putea include comanda de deschidere a bazei de date, iar în codul cleanup, închiderea acesteia.

Pentru scrierea acestor proceduri va trebui invocată opțiunea View→General Options din meniul standard al VFP, după ce în prealabil a fost deschis Menu Designer. Se va invoca astfel dialogul General Options (vezi figura 12.12), care conține printre altele și două checkbox-uri Setup... și Cleanup... grupate în cadrul Menu Code. Bifarea fiecăreia dintre ele va determina deschiderea unei ferestre de editare cod (vezi figura 12.13).

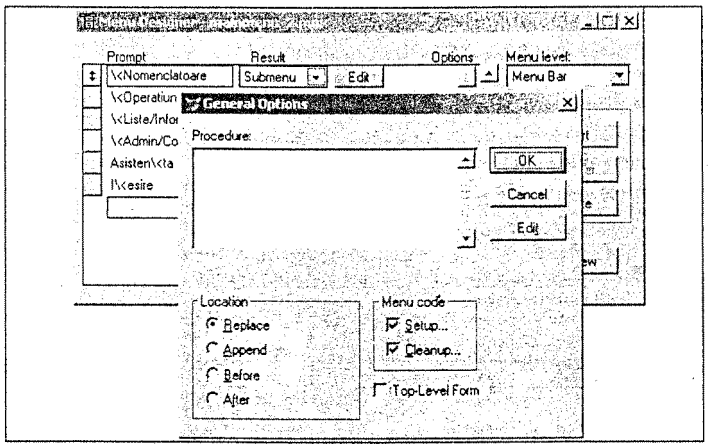


Figura 12.12. Fereastra de dialog General Options

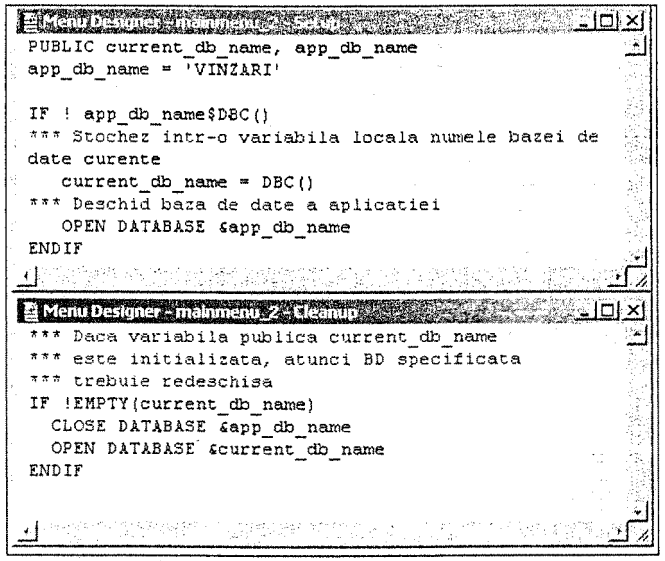


Figura 12.13. Ferestrele de editare a codului Setup și Cleanup invocate din dialogul General Options

12.1.7. Generarea unui meniu și câteva precizări sumare despre comenzile de definire a acestuia

Deși Menu Designer ușurează munca de definire a unui meniu atât de mult încât un utilizator novice poate defini fără dificultăți un meniu complet și destul de sofisticat fără a cunoaște pratic nici o comandă specifică, totuși interpretarea, chiar dacă nu foarte

aprofundată, a programului generat de Menu Designer necesită câteva precizări suplimentare în legătură cu acest subiect.

Generarea programului .MPR ca rezultat al specificațiilor din Menu Designer se realizează apelând opțiunea Menu→Generate din meniul standard VFP. Ca urmare, se va obține dialogul Generate Menu, unde se poate sugera (eventual) calea și numele fișierului-destinație, după cum este exemplificat în figura 12.14.

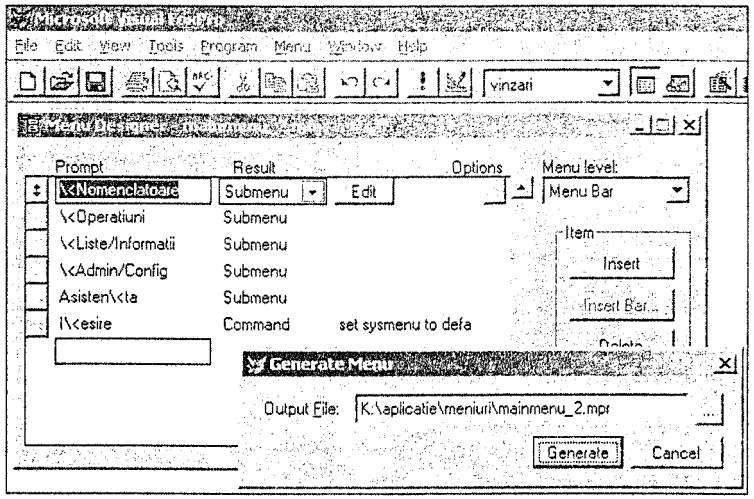


Figura 12.14. Generarea programului .MPR

Programul-sursă generat poate fi editat eventual printr-o comandă MODIFY COMMAND. În figura 12.15 este prezentat un fragment din acest program.

```
mainmenu.mpr
SET SYSMENU TO
SET SYSMENU AUTOMATIC

DEFINE PAD _0ix0ub8y8 OF _MSYSMENU PROMPT "\<Nomenclatoare" COLOR SCHEME 3 ;
    KEY ALT+N, ""
DEFINE PAD _0ix0ub8y9 OF _MSYSMENU PROMPT "\<Operatiuni" COLOR SCHEME 3 ;
    KEY ALT+O, ""
DEFINE PAD _0ix0ub8ya OF _MSYSMENU PROMPT "\<Liste/Informatii" COLOR SCHEME 3 ;
    KEY ALT+L, ""
DEFINE PAD _0ix0ub8yb OF _MSYSMENU PROMPT "\<Admin/Config" COLOR SCHEME 3 ;
    KEY ALT+A, ""
DEFINE PAD _0ix0ub8yc OF _MSYSMENU PROMPT "Asisten\<ta" COLOR SCHEME 3 ;
    KEY ALT+T, ""
DEFINE PAD _0ix0ub8yd OF _MSYSMENU PROMPT "I\<esire" COLOR SCHEME 3 ;
    KEY ALT+E, ""
ON PAD _0ix0ub8y8 OF _MSYSMENU ACTIVATE POPUP nomenclato
ON PAD _0ix0ub8y9 OF _MSYSMENU ACTIVATE POPUP operatiuni
ON PAD _0ix0ub8ya OF _MSYSMENU ACTIVATE POPUP listeinfor
ON PAD _0ix0ub8yb OF _MSYSMENU ACTIVATE POPUP adminconfi
ON PAD _0ix0ub8yc OF _MSYSMENU ACTIVATE POPUP asistenta
ON SELECTION PAD _0ix0ub8yd OF _MSYSMENU set sysmenu to default
```

Figura 12.15. Fragment din programul mainmenu.mpr

Principalele comenzi pe care le veți întâlni în acest program sunt:

- DEFINE PAD – pentru definirea principalelor opțiuni din meniul principal;
- ON PAD și ON SELECTION PAD – pentru definirea acțiunilor corespunzătoare opțiunilor din meniul principal (invocarea unor submeniuri, comenzi sau programe);
- DEFINE POPUP – pentru definirea submeniurilor corespunzătoare opțiunilor din meniul principal;
- ON BAR și ON SELECTION BAR – pentru definirea acțiunilor corespunzătoare opțiunilor fiecărui submeniu (popup) definit prin DEFINE POPUP.

Tot în acest program se vor regăsi și procedurile corespunzătoare codului-sursă definit la nivelul meniului principal, submeniurilor sau opțiunilor individuale.

12.2. Simularea unui meniu vertical și crearea barelor de instrumente

Mulți dezvoltatori se pot declara totuși nemulțumiți de posibilitățile de modelare a modului de prezentare a aplicației (look&feel-ul) oferite de instrumentul Menu Designer.

În continuare, vom prezenta două modalități de îmbunătățire a prezentării aplicației.

12.2.1. Definirea unui meniu de pornire vertical

Să facem pentru început o distincție: când ne referim în acest paragraf la un meniu vertical nu ne propunem crearea unui meniu popup (lucru perfect realizabil cu Menu Designer). De fapt, încercăm să construim o structură de interfață specială care în final va avea aspectul unui meniu vertical. Concret:

- vom crea o bară de stare care, pe lângă eventualele informații despre starea curentă a aplicației, va găzdui și un buton (asemănător butonului START al Windows-ului) care va lansa o bară asemănătoare unui meniu vertical;
- opțiunile din bara dedicată meniului vertical vor fi formate în fapt din butoane de comandă formatate specific și care vor ține locul opțiunilor din meniul principal descris în subcapitolul anterior. Fiecare dintre acestea va lansa un meniu popup care va avea funcția submeniului similar din meniul anterior creat cu Menu Designer;
- funcțiile opțiunilor individuale din meniurile popup vor respecta specificațiile din tabelul 12.1.

Pentru a realiza această structură de interfață, vom folosi elemente de programare OO pentru organizarea mai eficientă a procesului de dezvoltare, dar și fiindcă vom utiliza ca bază de pornire o clasă specifică din bibliotecile mediului vizual VFP.

Procesul de dezvoltare îl putem separa în două etape: (1) crearea claselor pe care se vor baza elementele specifice interfeței descrise mai sus; și (2) crearea setului de formulare în care se vor instanția aceste clase.

Crearea claselor specifice

În *primul pas* din această etapă ne vom ocupa de *clasa de proveniență a barei de stare*. Specificul barei de stare decurge din următoarele caracteristici:

- dimensiunea;
- plasamentul pe ecran;
- obiectele prin care vor fi afişate eventualele mesaje sau informaţii despre starea curentă a aplicaţiei.

Trebuie precizat că bara de stare va avea doar un rol de element de interfaţă găzduind eventual alte obiecte de acest gen pentru, de exemplu, afişarea unor mesaje specifice. Ea nu va avea nici o funcţie legată de monitorizarea mediului intern şi culegerea respectivelor informaţii relevante.

În acest scop, vom crea o clasă pe care o vom denumi `StatusBar` şi care, la rândul ei, va subclasa clasa `Toolbar` a mediului vizual VFP, după cum se poate observa din figura 12.16, în care este prezentat dialogul `New Class` obţinut prin apelarea opţiunii `File→New→Class` din meniul standard VFP.

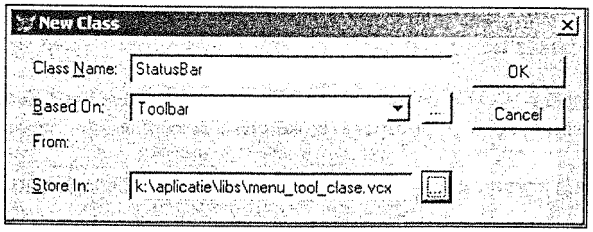


Figura 12.16. Crearea clasei `StatusBar` pe baza clasei `Toolbar`

Proprietăţile personalizate pentru clasa `StatusBar` se referă la dimensiune (`Height` şi `Width`) şi la plasament (`Left`, `Top`) – vezi figura 12.17.

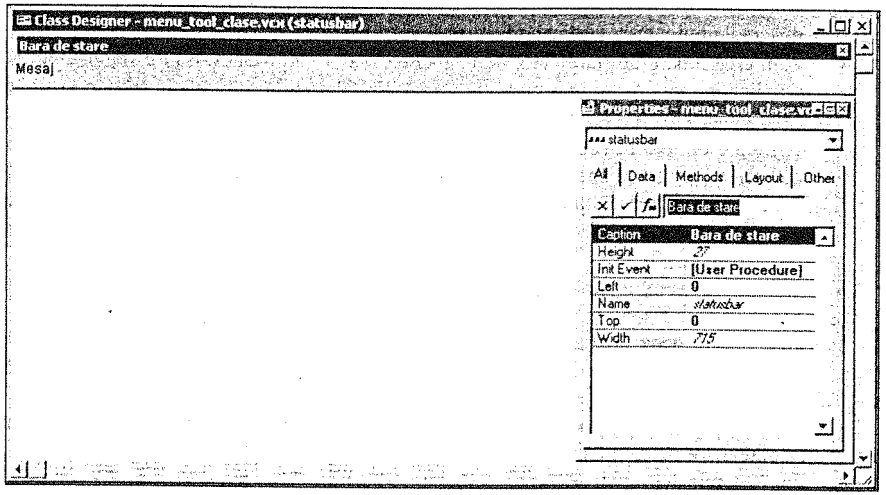


Figura 12.17. Fereastra de definire a clasei `StatusBar`

De asemenea, tot în figura 12.17 se observă şi includerea unui obiect de tip `label` cu numele `lblMesaj`, care va găzdui eventualele mesaje ce se doresc a fi afişate pe bara de stare.

O caracteristică esenţială a barei de stare este locul de „ancorare" în fereastra principală a aplicaţiei. În acest sens, avem la dispoziţie metoda `Dock` din clasa de bază `Toolbar`, care primeşte un parametru ce determină poziţionarea barei de stare. Astfel, valoarea:

- `-1` sau `TOOL_NOTDOCKED` (din `FoxPo.h`) înseamnă neancorarea în nici o margine (altfel spus, bara de stare este flotantă);
- 0 sau `TOOL_TOP` înseamnă ancorarea în marginea superioară a ferestrei VFP;
- 1 sau `TOOL_LEFT` înseamnă ancorarea în marginea stângă a ferestrei VFP;
- 2 sau `TOOL_RIGHT` înseamnă ancorarea în marginea dreaptă a ferestrei VFP;
- 3 sau `TOOL_BOTTOM` înseamnă ancorarea în marginea inferioară a ferestrei VFP.

Prin urmare, determinarea plasamentului barei de stare poate fi determinat prin evenimentul `Init` prezentat în figura 12.18.

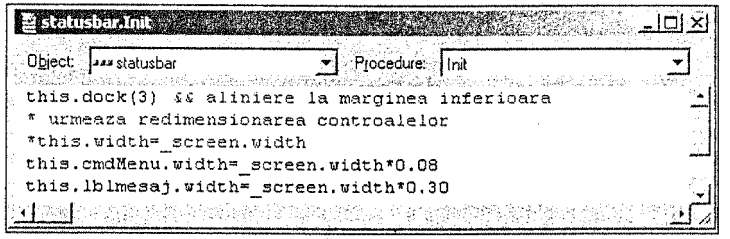


Figura 12.18. Evenimentul `Init` asociat clasei `StatusBar`

În *pasul doi* din această etapă vom crea *clasa care va sta la baza butonului de start al meniului vertical şi la baza butoanelor care vor forma opţiunile de bază ale acestui meniu.*

Această clasă o vom denumi `MenuToolBtn` şi va fi subclasată din clasa de bază `CommandButton` a mediului vizual VFP. Proprietăţile redefinite pentru această clasă se referă la (vezi şi figura 12.19):

- dimensionare (`Height` şi `Width`);
- formatul fontului pentru afişarea textului care etichetează butonul (`FontBold`, `FontSize` şi `ForeColor`);
- efectul 3D de afişare (`SpecialEffect`).

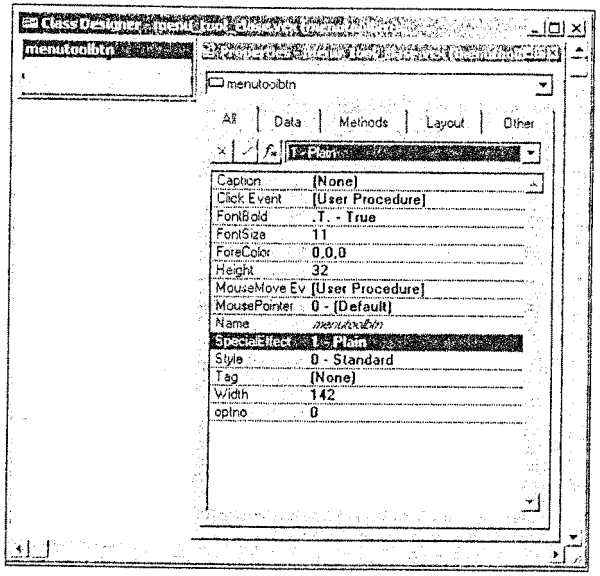


Figura 12.19. Fereastra de editare a clasei `MenuButtonTlb`

În plus, am adăugat o proprietate suplimentară – `OptNo` – prin care să poată fi identificabil mai uşor butonul în setul de opţiuni care va forma meniul vertical. Adăugarea unei proprietăţi specifice la nivelul unui obiect se realizează prin intermediul opţiunii `Class→NewProperty`, apelabilă din mediul standard VFP când este deschis instrumentul `ClassDesigner` (vezi şi capitolul 11).

Specificul comportamentului butoanelor-opţiuni care vor instanţia această clasă este determinat de două caracteristici:

- lansarea meniului popup corespunzător;
- modificarea aspectului vizual (mai exact, culoarea fontului) atunci când butonul mouse-ului trece pe deasupra.

Pentru a realiza acest lucru, am scris câteva linii specifice de cod în evenimentele `Click` şi `MouseMove` prezentate în figura 12.20.

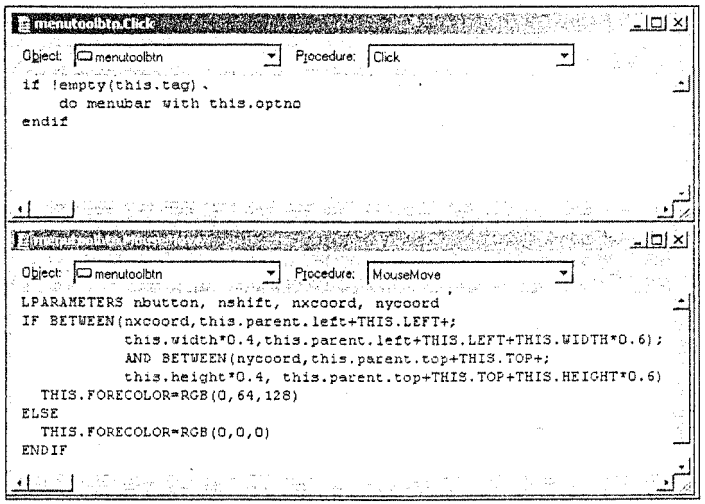


Figura 12.20. Codul-sursă asociat evenimentelor `Click` şi `MouseMove`

Meniurilor popup care vor fi lansate din butoanele-opţiuni ale meniului vertical sunt definite într-o procedură separată – `menubar.prg`. Aceasta este apelabilă printr-un parametru specific, care determină meniul popup ce urmează a fi activat. Numărul meniului popup din această procedură corespunde numărului butonului-opţiune (determinat de proprietatea `OptNo`) aparţinând meniului vertical (vezi listingul 12.2).

Listing 12.2. Procedura `menubar.prg`

```
PARAMETER pmenu

DEFINE POPUP shortcut shortcut RELATIVE FROM MROW(),MCOL()
DO CASE
CASE pmenu = 1
DEFINE BAR 1 OF shortcut PROMPT "Per\<soane"
DEFINE BAR 2 OF shortcut PROMPT "\<Produse"
DEFINE BAR 3 OF shortcut PROMPT "\<Clienti"
ON SELECTION BAR 1 OF shortcut DO menuoption with 11
ON SELECTION BAR 2 OF shortcut DO menuoption with 12

CASE pmenu = 2
DEFINE BAR 1 OF shortcut PROMPT "\<Facturari"
DEFINE BAR 2 OF shortcut PROMPT "\<Incasari"
ON SELECTION BAR 1 OF shortcut DO menuoption with 21

CASE pmenu = 3
DEFINE BAR 1 OF shortcut PROMPT "Lista \<clientilor"
DEFINE BAR 2 OF shortcut PROMPT "Lista \<facturilor"
DEFINE BAR 3 OF shortcut PROMPT "Lista \<vinzari clienti/produse"
DEFINE BAR 4 OF shortcut PROMPT "\-"
DEFINE BAR 5 OF shortcut PROMPT "\<Informatii despre facturi..."
ON SELECTION BAR 1 OF shortcut DO menuoption with 31
ON SELECTION BAR 2 OF shortcut DO menuoption with 32
ON SELECTION BAR 3 OF shortcut DO menuoption with 33
```

```
ON SELECTION BAR 5 OF shortcut DO menuoption with 34

CASE pmenu = 4
DEFINE BAR 1 OF shortcut PROMPT "\<Administrare BD"
DEFINE BAR 2 OF shortcut PROMPT "Configurare \<retea"
DEFINE BAR 3 OF shortcut PROMPT "Configurare \<imprimanta"
ON BAR 1 OF shortcut ACTIVATE POPUP admin
ON BAR 3 OF shortcut ACTIVATE POPUP imprim

DEFINE POPUP admin shortcut RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR 1 OF admin PROMPT "Refacere indecsi"
DEFINE BAR 2 OF admin PROMPT "Verific restrictii"
DEFINE BAR 3 OF admin PROMPT "Copie de siguranta BD"
DEFINE BAR 4 OF admin PROMPT "Restaurare BD"
ON SELECTION BAR 1 OF admin DO menuoption with 411
ON SELECTION BAR 2 OF admin DO menuoption with 412

DEFINE POPUP imprim shortcut RELATIVE SHADOW COLOR SCHEME 4
DEFINE BAR _mfi_sysprint OF imprim PROMPT "\<Imprimanta"
DEFINE BAR _MFI_PGSET OF imprim PROMPT "\<Pagina"

CASE pmenu = 5

DEFINE BAR 1 OF shortcut PROMPT "\<Continut si index...."
DEFINE BAR 2 OF shortcut PROMPT "\-"
DEFINE BAR 3 OF shortcut PROMPT "\<Despre"
ON SELECTION BAR 1 OF shortcut HELP

ENDCASE

ACTIVATE POPUP shortcut

**************************************************
procedure menuoption
parameter opt_
do hide_menubar

do case
case opt_ = 11  && Nomenclatoare\Persoane
 DO FORM frmPersoane
case opt_ = 12  && Nomenclatoare\Produse
 DO FORM frmProduse
case opt_ = 21  && Operatiuni\Facturari
 DO FORM FrmLiniiFact
case opt_ = 31  && Liste\Lista clientilor
 REPORT FORM rep_clienti PREVIEW
case opt_ = 32  && Liste\Lista facturilor
 REPORT FORM rep_facturi_clienti PREVIEW
case opt_ = 33  && Liste\Lista vinzari
 REPORT FORM rap_dinamic PREVIEW
case opt_ = 34  && Liste\Informatii facturi
 DO FORM \aplicatie\FORMS\frmtree_list.scx
case opt_ = 411  && Adminconfig\AdminBD\Refacere indecsi
 DO refacere_indecsi
case opt_ = 412  && Adminconfig\AdminBD\Verificare RR
 DO restrictii_referentiale_macro
endcase
```



```
do activate_menu
**************************************************
procedure hide_menubar
FOR k=1 TO _SCREEN.FORMCOUNT
  IF LOWER(_SCREEN.FORMS(k).NAME)='menubar'
    EXIT
  ENDIF
ENDFOR
_SCREEN.FORMS(k).HIDE
**************************************************
procedure activate_menu
FOR k=1 TO _SCREEN.FORMCOUNT
  IF LOWER(_SCREEN.FORMS(k).NAME)='statusbar'
    EXIT
  ENDIF
ENDFOR
_SCREEN.FORMS(k).cmdMenu.ENABLED=.T.
```

În fine, în *pasul trei* din această etapă vom crea *clasa-suport pentru bara de meniu pe care vor fi plasate butoanele-opțiuni* ce vor forma opțiunile principale ale meniului vertical.

În legătură cu această clasă (prezentată în figura 12.21), trebuie să precizăm că are la bază tot clasa `Toolbar`, iar rolul ei este de a găzdui butoanele-opțiuni care vor forma meniul vertical. Caracteristicile fizice ale acesteia sunt determinate de evenimentul `Init`, în care dimensiunea ei este determinată relativ la înălțimea ferestrei VFP, iar plasamentul este determinat prin metoda `Dock`, care o ancorează în marginea din stânga.

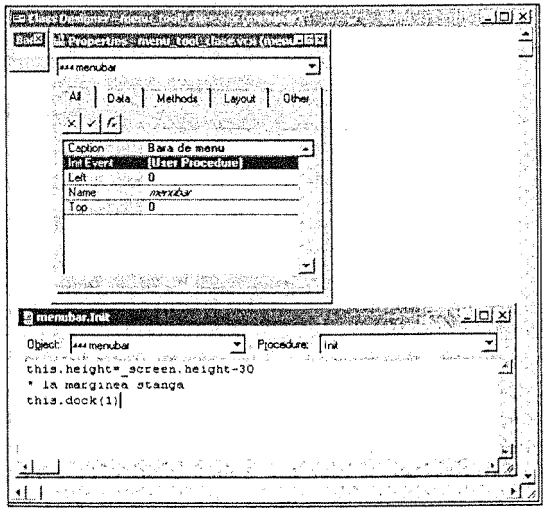


Figura 12.21. Clasa `MenuBar`

Crearea setului de formulare care va instanția clasele definite anterior

Subclasarea claselor definite în etapa anterioară se realizează în felul următor:

1. Se creează un nou formular.

2. În bara de instrumente se invocă obiectele din biblioteca în care au fost salvate clasele-suport. Acest lucru se realizează astfel (vezi figura 12.22 şi capitolul 11):
- Din bara de instrumente a `Form Designer`-ului se invocă butonul `View Classes`, iar din meniul care apare se apelează opţiunea `Add...`
- În fereastra `Open` se localizează biblioteca în care se află clasele-suport (fişierul `menu_tool_clase.vcx`).
- Ca urmare, în lista din meniul `Add...` invocat anterior va apărea şi biblioteca noastră, iar ca urmare a selectării opţiunii corespunzătoare, bara de instrumente a instrumentului `Form Designer` se transformă în aşa fel încât va conţine o serie de butoane din care se pot invoca clasele-suport.

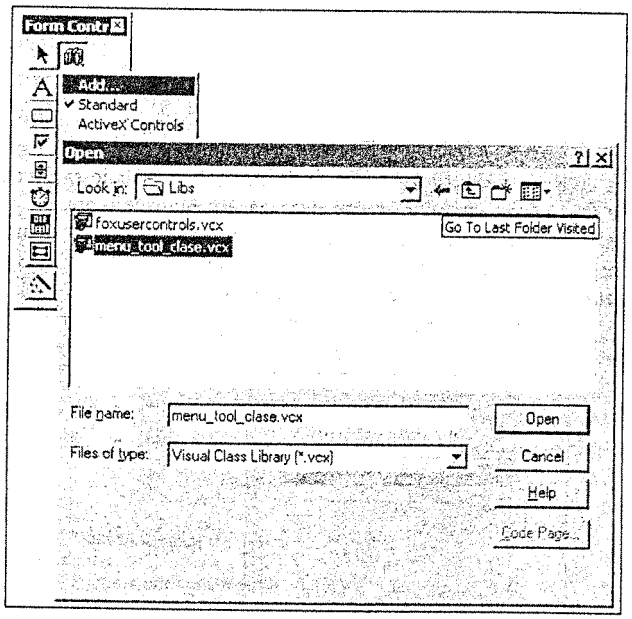


Figura 12.22. Invocarea bibliotecii care conţine clasele-suport

3. Prin butoanele specifice claselor-suport care au fost aduse în bara de instrumente `Form Control Toolbar` sunt „trase" în fereastra `Form Designer` clasele `StatusBar` şi `MenuBar`. Ca urmare, se va forma un set de formulare ce va cuprinde şi cele două formulare bazate pe clasele-suport definite în etapa anterioară.
4. Prin opţiunea `Form→Remove Form` din meniul VFP se îndepărtează formularul iniţial.

După care:
- pe bara de stare este plasat un buton etichetat `Menu`, subclasat din clasa `MenuToolBtn`;
- pe bara de meniu se plasează şase butoane-opţiune (subclasate tot din clasa `MenuToolBtn`) etichetate şi grupate corespunzător. Gruparea se poate realiza folosind un obiect de tip separator între butoanele ce marchează sfârşitul unui grup şi începutul celuilalt;
- în bara de meniu se poate include şi un obiect de tip `Image` în care să fie afişat un *logo* specific.

Până acum, rezultatul ar trebui să arate după cum este ilustrat în figura 12.23.

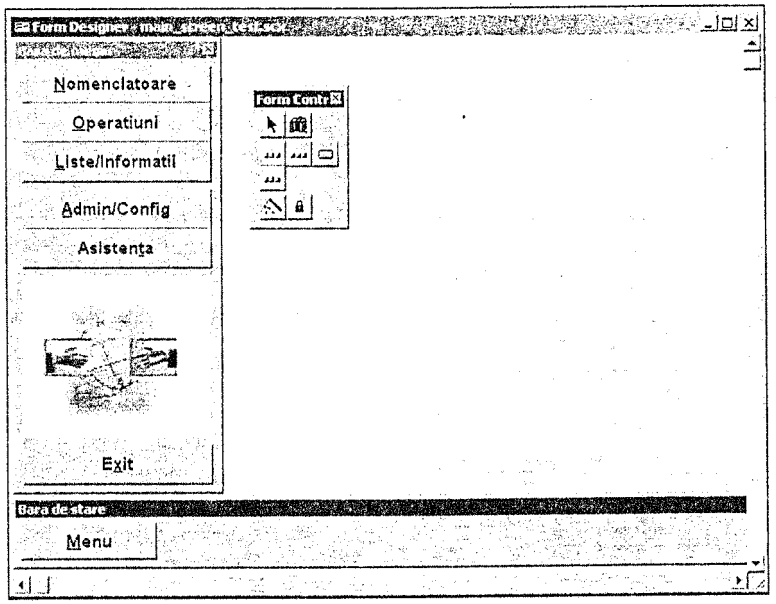


Figura 12.23. Setul de formulare realizat în `Form Designer`

În plus, am mai adăugat câteva elemente legate de comportament. Astfel:
- în momentul în care se va trece cursorul mouse-ului peste un buton-opţiune, pe bara de stare va apărea un mesaj explicativ referitor la opţiunea curentă. Acest comportament se obţine schimbând proprietatea `Caption` a obiectului `lblMesaj` de pe bara de stare printr-o linie de cod plasată într-un eveniment `MouseMove`;
- în momentul în care se execută click pe butonul etichetat `Menu` de pe bara de stare, este afişată bara de meniu. Acest comportament se obţine prin metoda `Show` a obiectului prin care se afişează bara de meniu.

Codul-sursă al evenimentelor invocate mai sus este prezentat în figura 12.24.

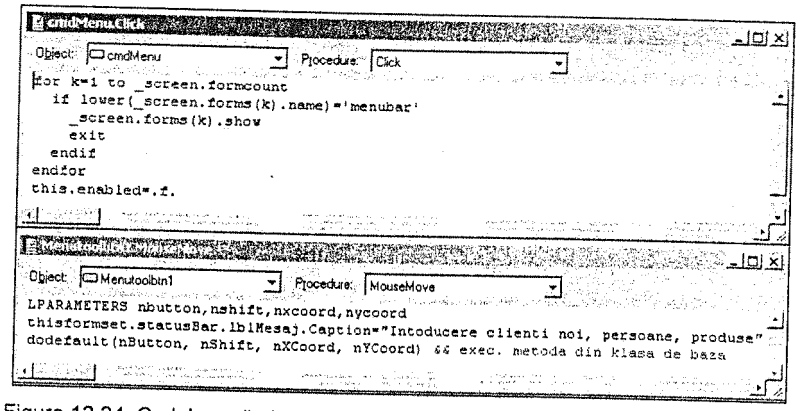


Figura 12.24. Codul-sursă al evenimentului `MouseMove` al butoanelor-opțiuni și al evenimentului `click` al butonului care afișează bara de meniu

În figura 12.25 este prezentat meniul vertical obținut prin lansarea setului de formulare la care am trudit până acum.

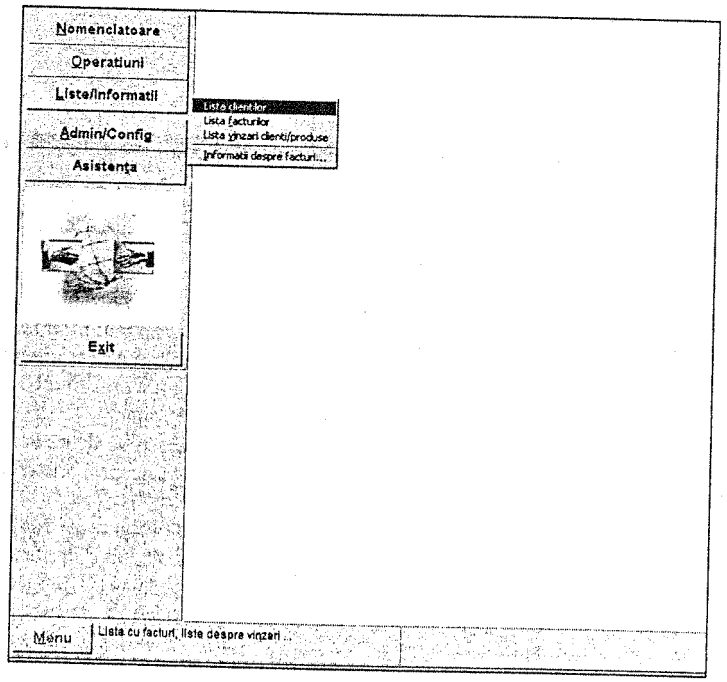


Figura 12.25. Meniul vertical obținut la execuție

12.2.2. Crearea unei bare de instrumente pentru aplicație

Aplicațiile care se doresc realizate cu un aspect cât mai profesionist sunt însoțite obligatoriu și de o bară de instrumente din care să se poată avea acces direct la principalele funcții ale sistemului.

Procesul de creare a unei bare de instrumente poate fi structurat în următoarele etape:

- identificarea principalelor funcții care vor fi accesibile din bara de instrumente a aplicației;
- crearea/desemnarea imaginilor relevante care vor defini butoanele asociate funcțiilor definite anterior;
- crearea unui element de interfață tip bară de instrumente subclasat din clasa de bază `Toolbar` a VFP;
- așezarea pe obiectul tip `toolbar` creat anterior a butoanelor ce vor forma bara de instrumente;
- specificarea imaginilor care vor fi aplicate pe fiecare buton din bara de instrumente;
- asocierea codului prin care se lansează procedura/formularul/raportul asociat fiecărui buton din `Toolbar`.

Să luăm în considerare, pentru exemplul nostru, următoarele funcții ce vor fi accesibile din `Toolbar`:

- crearea/modificarea datelor despre clienți;
- emiterea de noi facturi;
- consemnarea documentelor de încasare;
- afișarea unui raport privind vânzările;
- afișarea unui raport privind facturile;
- configurarea/pregătirea imprimantei pentru listare;
- crearea copiei de siguranță pentru baza de date;
- invocarea sistemului de asistență;
- părăsirea aplicației.

Imaginile ce vor fi afișate pe butoanele barei de instrumente trebuie create sau redefinite astfel încât să respecte dimensiunea butoanelor. Pentru o mai bună organizare, ele ar trebui plasate într-un director dedicat, după cum se ilustrează în figura 12.26. Acestea pot avea extensia `.BMP`, dar și `.ICO` sau `.JPG`.

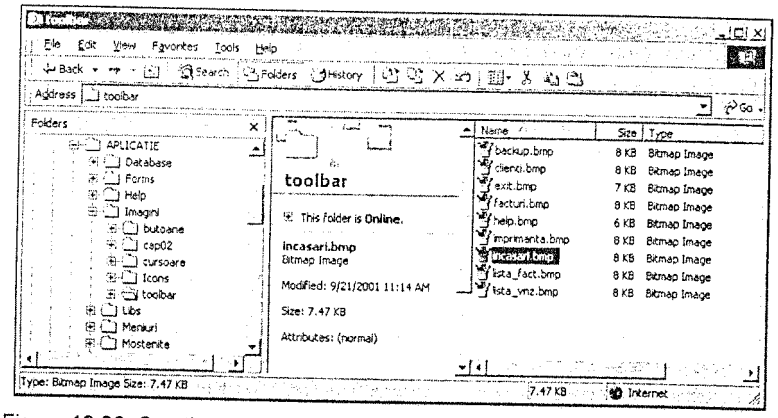


Figura 12.26. Creați un director separat dedicat exclusiv imaginilor/iconogramelor ce vor fi afișate în bara de instrumente

În continuare, pentru a simplifica procesul de dezvoltare, revenim în biblioteca `Menu_tool_clase.vcx` (creată pentru clasele-suport ale meniului vertical), în care introducem o nouă clasă având ca părinte clasa `Toolbar` a sistemului VFP și pe care o vom numi `apptoolbar`.

De asemenea, revenim în setul de formulare (`main_screen.scx`) creat anterior și adăugăm un nou obiect prin subclasarea clasei `Apptoolbar`, după cum se ilustrează în figura 12.27.

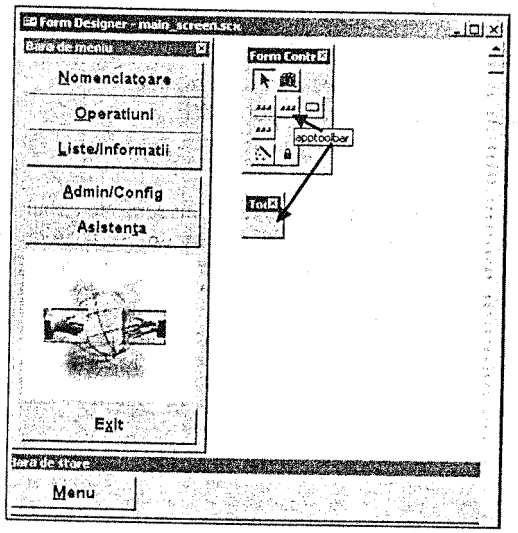


Figura 12.27. Subclasarea clasei `Apptoolbar` în setul de formulare `Main_screen`

Se plasează apoi nouă obiecte de tip `CommandButton` (corespunzătoare funcțiilor identificate mai devreme) pe noul formular obținut din subclasarea clasei `Apptoolbar`. De asemenea, pentru fiecare buton se va specifica, prin intermediul proprietății `Picture`, fișierul care conține imaginea sau iconograma care îi va fi asociată, obținându-se astfel rezultatul din figura 12.28.

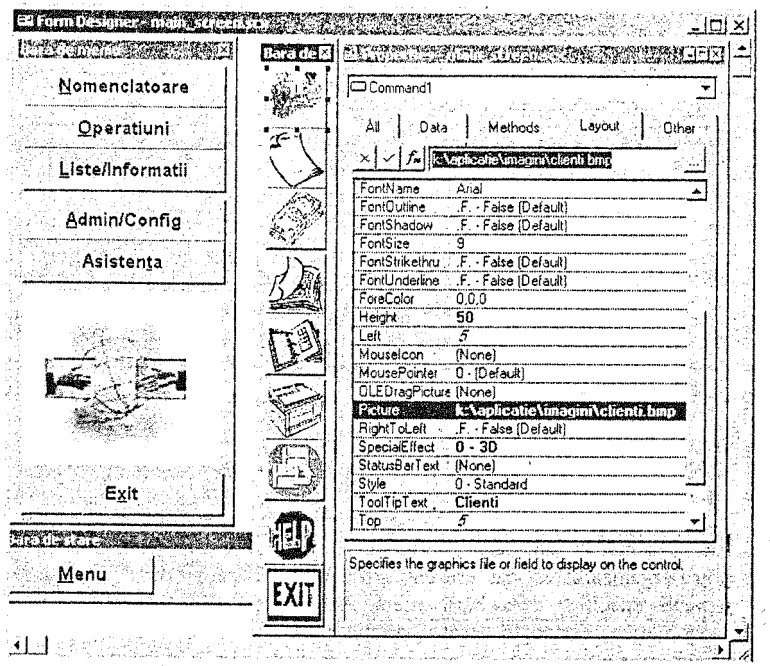


Figura 12.28. Asocierea imaginilor/iconogramelor pentru butoanele care formează bara de instrumente

În fine, tot pentru fiecare buton se va scrie în evenimentul `click` instrucțiunea prin care se va lansa în execuție programul, formularul sau raportul ce i-a fost desemnat. De exemplu, pentru butonul corespunzător creării/editării informațiilor despre clienți se poate specifica formularul `FrmClienti.scx` specificând în codul-sursă asociat evenimentului `click` următoarea comandă:

```
DO FORM frmclienti.scx
```

La execuție, selectarea acestui buton va determina rezultatul prezentat în figura 12.29.

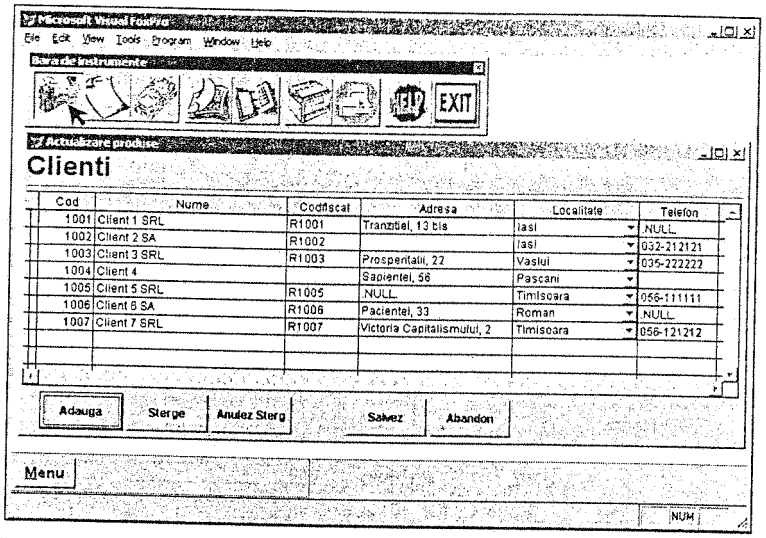


Figura 12.29. Lansarea formularului `Frmclienti.scx` din bara de instrumente

Crearea unui meniu bine structurat şi a unei bare de instrumente prin care să poată fi accesate principalele funcţii ale aplicaţiei se pot dovedi elemente esenţiale în obţinerea succesului privind gradul de acceptabilitate al sistemului la nivelul utilizatorilor finali.

„Unealta" cea mai la îndemână este `Menu Designer`, care oferă o gamă largă de opţiuni pentru organizarea cât mai eficientă şi flexibilă a meniului principal. Folosind însă câteva elemente specifice dezvoltării orientate-obiect, se poate obţine o structură de interfaţă ceva mai spectaculoasă decât meniurile construite cu `Menu Designer` şi care poate avea un impact mai mare în ceea ce priveşte aderenţa utilizatorilor finali la noul sistem.

Capitolul 13

Opţiuni pentru dezvoltarea aplicaţiilor în reţea. Arhitectura *file-server*

Bazele de date au apărut din necesitatea partajării resurselor informaţionale între diferitele categorii de utilizatori dintr-o organizaţie, înlăturându-se – sau măcar diminuându-se – preluarea şi stocarea repetată a aceloraşi tranzacţii: facturi, încasări, plăţi etc. Majoritatea copleşitoare a aplicaţiilor economice instalate în organizaţii medii şi mari (deşi putem împărţi aceste două tipuri în mai multe categorii) vizează accesul mai multor utilizatori la aceleaşi date – lucrul în reţea.

Lucrurile sunt simple atunci când aplicaţiile utilizatorilor „citesc" o aceeaşi înregistrare, însă problemele se multiplică dacă mai mulţi operatori modifică simultan valori ale unor atribute aflate pe o aceeaşi linie într-o tabelă, sau adaugă simultan înregistrări în tabele în care cheia primară este o cheie-surogat generată automat, sau unul şterge şi altul modifică aceeaşi înregistrare ş.a.m.d.

Prin urmare, opţiunile pentru lucrul în reţea reprezintă un element esenţial în alegerea unui SGBD ca mediu de dezvoltare a aplicaţiilor ce vor rula (vor fi exploatate) în medii multiutilizator. În acest capitol ne vom ocupa de cea mai simplă arhitectură a aplicaţiilor multiutilizator în VFP – *file-server*.

13.1. Arhitectura *file-server*

Nu există o traducere satisfăcătoare a sintagmei, aşa că vom importa lingvistic „originalul" din engleză. În această configuraţie, schematizată în figura 13.1, baza de date este rezidentă pe un calculator cu un hard-disc mai „încăpător" şi mai rapid (de ex. SCSI).

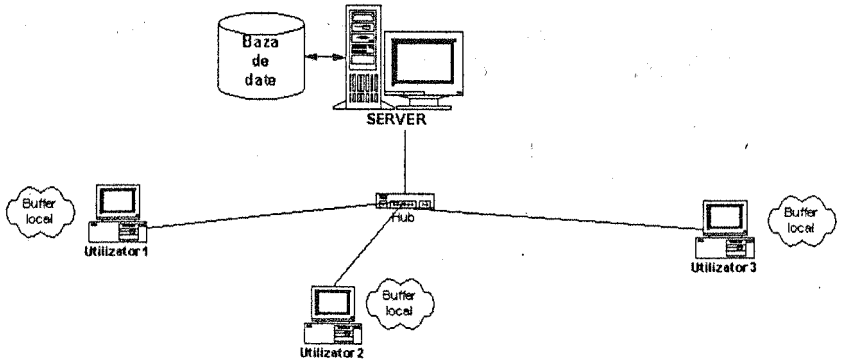


Figura 13.1. O arhitectură *file-server*

Utilizatorii, aflaţi fiecare la biroul său, au acces la bază fie prin intermediul aplicaţiilor (cazul tipic al utilizatorilor finali fără cunoştinţe supărătoare de informatică), fie prin comenzi lansate din VFP instalat pe staţia proprie.

De obicei, serverul nu trebuie să fie foarte puternic, prin comparaţie cu staţiile utilizatorilor, deoarece în această arhitectură prelucrările au loc local, în bufferul (memoria RAM) fiecărei staţii. Spre exemplificare, presupunem că utilizatorul 2 introduce din fereastra de comenzi următoarea frază SELECT:

```
SELECT * FROM liniifact WHERE nrfact = 5678
```

În bufferul local al staţiei 2 se încarcă, de pe serverul de fişiere, toate liniile tabelei LINIIFACT, linii din care se vor selecta cele care îndeplinesc condiţia din WHERE. Dacă în reţea sunt 50 de utilizatori care lansează simultan fraza de mai sus, iar LINIIFACT are câteva milioane de înregistrări, putem presupune că reţeaua are toate şansele să se „sufoce".

Dotarea serverului de fişiere cu un hard-disc rapid înseamnă onorarea într-un termen scurt a cererilor de acces la bază primite din partea staţiilor şi poate avea un aport important în fluidizarea fluxului de date în reţea.

Mecanismul de citire şi transfer al datelor între bufferele locale şi „instanţa" principală a bazei aflate pe server este transparent pentru utilizator. Important este însă să cunoaştem cele mai importante scenarii de conflicte ce ar putea să apară în utilizare şi ce opţiuni VFP sunt mai indicate în buna derulare a aplicaţiei şi păstrarea integrităţii şi coerenţei bazei.

În practică, cele discutate pe parcursul acestui capitol sunt valabile într-o reţea de tip peer-to-peer, în care nu există un *file-server* Windows, fiecare staţie comunicând cu celelalte „de la egal la egal". Pentru VFP acest lucru este secundar, deoarece, oricum, dintre staţii se alege una care va găzdui baza de date şi va juca rolul de server al fişierelor aplicaţiei.

Utilizarea unui server, fie şi numai pentru fişiere, este cvasigeneralizată deoarece, în afara unui hard-disc şi conectică (placă de reţea) rapide, serverul poate fi depozitat într-o încăpere separată şi protejat fizic mai bine. O altă practică este de a dota serverul cu un dispozitiv de salvare periodică a datelor din bază – unitate de bandă (casetă) magnetică sau inscriptoare de CD-uri.

13.2. Deschiderea bazei de date şi tabelelor pentru lucru în reţea. Moduri de blocare

Baza de date şi tabelele pot fi deschise în două moduri, exclusiv (EXCLUSIVE) şi partajat (SHARED). Prin deschiderea exclusivă, baza de date/tabela devine inaccesibilă următorilor utilizatori care ar încerca o nouă deschidere de pe staţiile lor şi care ar avea parte de mesajul de eroare File access is denied (1705). Aşa încât lucrul în reţea este posibil numai prin comenzile:

```
OPEN DATABASE vinzari SHARED
```

şi (de exemplu)

```
USE judete IN 0 SHARED
```


Dacă în comenzile OPEN DATABASE sau USE nu se specifică modul de deschidere, implicit acesta este cel setat prin meniu: Tools→Options→Data şi bifarea/debifarea opţiunii Open exclusive sau prin comanda SET EXCLUSIVE.

Pentru a verifica dacă, în sesiunea locală, baza de date sau o tabelă au fost deschise exclusiv sau partajat, se poate folosi funcţia ISEXCLUSIVE():

```
? ISEXCLUSIVE('judete')
```

care întoarce, în cazul nostru, .T. şi este echivalentă cu

```
? ISEXCLUSIVE('judete', 1)
```

Pentru baza de date se scrie:

```
? ISEXCLUSIVE('vinzari', 2)
```

Atenţie însă, cu această funcţie nu se poate verifica dacă baza de date sau tabela a fost deschisă deja de alt utilizator (altă sesiune VFP pentru baza de date respectivă) ! Procedura din listingul 13.1 ilustrează modalitatea în care poate fi rezolvată această problemă.

Listing 13.1. Semnalizarea imposibilităţii deschiderii exclusive unei tabele în reţea

```
* se testeaza daca tabela nu e deja deschisa exclusiv
IF USED('clienti') AND ISEXCLUSIVE('clienti')
    * e-n regula (in engleza - OK)
ELSE
    IF USED('clienti')
        SELECT clienti
        USE
    ENDIF

    * se "inhiba" afisarea pe ecran a mesajului erorii 1705
    * ( File access is denied)
    ON ERROR aaa = 1

    * se incearca deschiderea exclusiva a tabelei
    USE clienti IN 0 EXCLUSIVE

    * daca nu s-a reusit, tabela ramine inchisa pt. sesiunea curenta
    IF !USED('clienti')
        MESSAGEBOX('Tabela CLIENTI n-a putut fi deschisa exclusiv !')
    ENDIF

    * se reia afisarea mesajelor de eroare
    ON ERROR

ENDIF
```

13.2.1. Deschiderea partajabilă a bazei de date şi tabelelor

Fireşte, pentru o aplicaţie în reţea interesează deschiderea partajabilă a bazei şi tabelelor. Pentru un plus de funcţionalitate, pornim de la realitatea că, pe o altă staţie, o tabelă a fost deschisă în mod partajabil, dar este blocată temporar (de exemplu, printr-o funcţie FLOCK()). Listingul 13.2 conţine programul care încearcă de trei ori, la interval de cinci secunde, să deschidă SHARED tabela CLIENTI. Mesajul de avertizare este afişat abia după eşuarea celei de-a treia tentative de deschidere. Pentru un plus de atractivitate, în colţul din dreapta-sus este afişat un mesaj ce indică a câta tentativă de deschidere este.

Listing 13.2. Trei tentative de deschidere SHARED a tabelei CLIENTI

```
succes_ = .F.

FOR i = 1 TO 3
  WAIT "Tentativa nr. "+STR(i,1)+" de deschidere partajata a tabelei CLIENTI" ;
       WINDOW NOCLEAR TIMEOUT 5
  succes_ = incercare_deschidere()
  IF succes_
       EXIT
  ENDIF
ENDFOR
IF !succes_
  MESSAGEBOX('Tabela CLIENTI n-a putut fi deschisa partajabil '+CHR(13)+;
       ' nici dupa trei tentative (15)secunde !')
ENDIF
WAIT CLEAR

RETURN

FUNCTION incercare_deschidere
* se testeaza daca tabela nu e deja deschisa partajat
IF USED('clienti') AND !ISEXCLUSIVE('clienti')
  * e-n regula
ELSE
  IF USED('clienti')  && tabela e deja deschisa exclusiv
       SELECT clienti
       USE
  ENDIF

  * se "inhiba" afisarea pe ecran a mesajului erorii 1705
  * ( File access is denied)
  ON ERROR aaa = 1

  * se incearca deschiderea partajata a tabelei
  USE clienti IN 0 SHARED

  * se reia afisarea mesajelor de eroare
  ON ERROR

  * nu s-a reusit deschiderea; se returneaza FALSE
  IF !USED('clienti')
       RETURN .F.
  ENDIF
ENDIF
* e-n regula ! tabela e deschisa SHARED
RETURN .T.
```

În acest moment ne putem gândi la modificarea procedurii stocate SETUP, în care să se verifice dacă, pe stația respectivă, pot fi deschise SHARED toate tabelele bazei. Dacă nu, aplicația nu trebuie să „pornească". Pentru fiecare tabelă sunt trei tentative de deschidere, din cinci în cinci secunde, înainte de afișarea mesajului de eroare. Iată noua variantă în listingul 13.3.

Listing 13.3. Varianta pentru rețea a procedurii stocate SETUP și a procedurilor conexe

```
*=============================================================
      PROCEDURE SETUP                    && varianta RETEA
*-------------------------------------------------------------
* se verifica daca baze de date e deschisa SHARED
IF DBUSED('vinzari') AND !ISEXCLUSIVE('vinzari',2)
  * e-n regula
ELSE
      MESSAGEBOX('Baza de date VINZARI nu este deschisa partajabil ')
*     se poate alege iesirea din aplicatie, chiar din VFP
*     QUIT
*     sau predarea controlului programumui principal
*     RETURN TO MASTER
ENDIF

* se incearca a se deschide SHARED toate tabelele BD
PUBLIC vTabele
DIME vTabele(5,1)
ADBOBJECTS(vTabele, "TABLE")
LOCAL i
FOR i = 1 TO ALEN(vTabele)
  tab_ = vTabele(i)
  * apelul functiei care incearca deschiderea SHARED a tabelei

  IF !deschidere_tabele (UPPER(ALLT(tab_)))
      MESSAGEBOX('Cu durere in glas, aplicatia nu poate fi lansata !')
*     se poate alege iesirea din aplicatie, chiar din VFP
      QUIT
*     sau predarea controlului programului principal
*     RETURN TO MASTER
  ENDIF

  * daca exista macar un index, primul (dintre indecsi) devine principal
  SELECT (tab_)
  IF TAGCOUNT() > 0
      SET ORDER TO TAG 1
  ENDIF
  PUBLIC trg_ins_&tab_, trg_upd_&tab_, trg_del_&tab_
ENDFOR

PUBLIC vRelatii, vCopii, vParinti, vTaguriCopil, vTaguriParinte
DIME vRelatii(4,4), vCopii(1), vParinti(1), vTaguriCopil(1), vTaguriParinte(1)
ADBOBJECTS(vRelatii, 'Relation')
* prima coloana din vRelatii - tabela COPIL
* a doua coloana din vRelatii - tabela PARINTE
* a treia coloana din vRelatii - index (tag) COPIL
* a patra coloana din vRelatii - index(tag) PARINTE
DIME vCopii(ALEN(vRelatii,1)), vParinti(ALEN(vRelatii,1)), ;
  vTaguriCopil(ALEN(vRelatii,1)), vTaguriParinte(ALEN(vRelatii,1))
FOR i = 1 TO ALEN(vRelatii,1)
  vCopii(i) = vRelatii(i,1)
  vParinti(i) = vRelatii(i,2)
  vTaguriCopil(i) = vRelatii(i,3)
  vTaguriParinte(i) = vRelatii(i,4)
ENDFOR
```

```
** la fiecare iesire din VFP se salveaza o copie a procedurilor stocate
ON SHUTDOWN DO salvare_proceduri_stocate

ENDPROC
*=============================================================================


PROCEDURE deschidere_tabele
PARAMETER tabela_
LOCAL i
succes_ = .F.
FOR i = 1 TO 3
  succes_ = incercare_deschidere((tabela_))
  IF succes_
     EXIT
  ELSE
     WAIT "Tentativa nr. "+STR(i,1)+" de deschidere partajata a tabelei "+tabela_ ;
     WINDOW NOCLEAR TIMEOUT 5
  ENDIF
ENDFOR
IF !succes_
  MESSAGEBOX('Tabela ' + tabela_+ ' n-a putut fi deschisa partajabil '+CHR(13)+;
     ' nici dupa trei tentative (15)secunde !')
  RETURN .F.
ENDIF
WAIT CLEAR
RETURN .T.
RETURN

*-----------------------------------------------------------------------------
FUNCTION incercare_deschidere
PARAMETER tabela_

* se testeaza daca tabela nu e deja deschisa partajat
IF USED((tabela_)) AND !ISEXCLUSIVE((tabela_))
  * e-n regula
ELSE
  IF USED((tabela_))  && tabela e deja deschisa exclusiv
     SELECT (tabela_)
     USE
  ENDIF
  * se "inhiba" afisarea pe ecran a mesajului erorii 1705
  * ( File access is denied)
  ON ERROR aaa = 1
  * se incearca deschiderea partajata a tabelei
  USE (tabela_) IN 0 SHARED
  * se reia afisarea mesajelor de eroare
  ON ERROR
  * nu s-a reusit deschiderea; se returneaza FALSE
  IF !USED((tabela_))
     RETURN .F.
  ENDIF
ENDIF
* e-n regula ! tabela e deschisa SHARED
RETURN .T.
```


13.2.2. Blocajul la nivel de tabelă

Bun, veți spune, dacă deschiderea exclusivă este atât de restrictivă, uităm de opțiunea `EXCLUSIVE` și ne legăm pe viață de `SHARED`. Din păcate, acest lucru nu este posibil, deoarece o serie de comenzi sunt operaționale numai dacă tabela a fost deschisă exclusiv:

- `ALTER TABLE` și `MODIFY STRUCTURE`;
- `INDEX` și `REINDEX`;
- `PACK` și `ZAP`;
- `INSERT [BLANK]` – comanda xBase (nu `INSERT SQL`).

În afara acestora, sunt o serie de comenzi care, deși tabela pe care operează este deschisă partajabil, blochează toate înregistrările, astfel încât toți ceilalți utilizatori pot vizualiza datele fără a modifica nici un atribut:

- `DELETE` cu opțiunile `ALL`, `FOR` sau `WHILE`;
- `RECALL` cu opțiunile `ALL`, `FOR` sau `WHILE`;
- `REPLACE` cu opțiunile `ALL`, `FOR` sau `WHILE`;
- `UPDATE` și `UPDATE SQL`;
- `TABLEUPDATE()` dacă a fost stabilit un buffering la nivel de tabelă (vezi paragraful următor).

Blocajul exclusiv al întregii tabele curente poate fi realizat și prin funcția `FLOCK()`, care întoarce `.T.` dacă operațiunea de blocare a reușit și `.F.` dacă tabela sau o înregistrare a sa a fost blocată de un alt utilizator. În mod obișnuit, funcția are o singură tentativă de „încuiere", însă numărul tentativelor sau perioada de încercări pot fi controlate prin comanda `SET REPROCESS`. O tabelă poate fi deblocată explicit prin comanda `UNLOCK` și implicit prin comenzile: `USE` (fără argumente – care închide tabela), `CLEAR ALL`, `CLOSE DATABASES`, `CLOSE TABLES` și la ieșirea din sesiunea VFP.

13.2.3. Blocajul antetului unei tabele

Blocajul la nivel de tabelă este cel mai sever, deoarece toți ceilalți utilizatori pot doar să citească înregistrările. Al doilea tip de „încuiere", ca severitate, este la nivel de antet al tabelei (*table header*), care permite modificarea înregistrărilor de la alte posturi, însă fără posibilitatea de a adăuga noi linii în tabela respectivă.

Următoarele comenzi „practică" acest tip de blocaj:

- `APPEND`;
- `APPEND BLANK`;
- `APPEND FROM`;
- `APPEND FROM ARRAY`;
- `INSERT SQL`.

Dacă doi utilizatori lansează simultan o asemenea comandă asupra aceleiași tabele, atunci al doilea va recepționa în plin ecran mesajul de eroare `File is in use by another`.

Blocarea explicită a antetului unei tabele poate fi realizată prin funcția RLOCK() sau LOCK(), în care valoarea argumentului care indică înregistrarea sau înregistrările de blocat este 0. De exemplu:

```
LOCK('clienti', 0)
```

sau

```
RLOCK ('facturi', 0).
```

„Descuierea" explicită a antetului se poate realiza prin comenzile:

- UNLOCK RECORD 0;
- UNLOCK;
- UNLOCK ALL.

13.2.4. Blocajul la nivel de înregistrare

Cel mai plauzibil scenariu de lucru în rețea este cel în care fiecare utilizator editează, la un moment dat, propria înregistrare. Pentru ca din momentul editării unei înregistrări (care este cel al primei modificări a valorii unui câmp) până la trecerea pe o altă înregistrare sau „comiterea" efectivă a modificărilor înregistrarea să-și păstreze coerența, majoritatea comenzilor de editare blochează înregistrarea vizată (blocarea are loc în momentul primei modificări a unui câmp):

- BLANK;
- BROWSE, CHANGE, EDIT;
- DELETE și DELETE RECORD;
- DELETE SQL;
- GATHER;
- MODIFY MEMO;
- REPLACE;
- SHOW GETS.

Acest tip de „încuiere" este cel mai puțin restrictiv și cel mai indicat în aplicațiile gândite a lucra în rețea. Dacă, spre exemplu, printr-o comandă BROWSE se încearcă modificarea unei înregistrări în curs de editare de alt utilizator (care a operat deja cel puțin o modificare în această înregistrare), se afișează mesajul de avertizare Attempting to lock... Press ESC to cancel, iar sesiunea de lucru este blocată. Deblocarea se realizează fie dacă celălalt utilizator termină modificările și „descuie" înregistrarea, fie dacă utilizatorul curent apasă tasa ESCape, semn că renunță.

În aplicații, mesajul de eroare ce poate să apară la încercarea de editare a unei înregistrări în curs de modificare în altă sesiune este Record is in use by another.

Din experiența noastră, comenzile de tipul celor de mai sus sunt destul de rezonabile în ceea ce privește mecanismul de blocare, în sensul că timpul de „încuiere" este foarte scurt, celelalte sesiuni nefiind incomodate de modificările de pe o stație. Cu totul altfel stau lucrurile atunci când se lucrează cu tranzacții. Să luăm în discuție procedura din listingul 13.4.

Listing 13.4. Testarea blocajului la nivel de înregistrare

```
SET REPROCESS TO 0
IF !USED('clienti')
  USE clienti IN 0 SHARED
ENDIF
*BEGIN TRANSACTION
SELECT CLIENTI
LOCATE FOR CodCl = 1003
nr_ = 1
sir1_ = "2"
FOR i = 1 TO 50000
  sir1_ = sir1_ + ALLT(STR(nr_))
  REPLACE Adresa WITH "Prosperitatii, "+sir1_
  nr_ = nr_ + 1
ENDFOR
*END TRANSACTION
RETURN
```

Lansată simultan de la două sau mai multe stații, procedura funcționează rezonabil, stațiile predându-și controlul la intervale scurte de timp, astfel încât nici una nu va fi blocată prea mult timp. Firește, dacă numărul stațiilor este mai mare, situația se complică. În schimb, dacă se „de-comentează" comenzile BEGIN TRANSACTION și END TRANSACTION, blocajul apare în toată splendoarea sa. Aceasta deoarece prima modificare apărută în cadrul tranzacției duce la „încuierea" înregistrării până la END TRANSACTION. Celelalte stații trebuie să aștepte ca procedura să se execute în totalitate, după care primesc, pe rând, controlul.

Funcțiile RLOCK() și LOCK() încearcă blocarea înregistrării curente un număr de ori/secunde stabilite prin comanda SET REPROCESS și întorc valoarea logică TRUE (.T.) dacă încuierea a fost reușită. Folosirea lor în programe prezintă avantaje precum: diminuarea și chiar înlăturarea timpilor de așteptare sau interblocajelor, semnalizarea imposibilității accesului la o anumită înregistrare, eliminarea riscului apariției de erori etc.

Listingul 13.5 prezintă o variantă evoluată a procedurii anterioare. Practic, de la stațiile de la care este lansat, programul modifică de 15.000 de ori valoarea adresei clientului 1003. După fiecare REPLACE, comanda UNLOCK descuie înregistrarea și afișează un mesaj de eroare, timp în care de pe celelalte stații se pot efectua modificări.

Listing 13.5. Folosirea funcție RLOCK()

```
SET REPROCESS TO 1

IF !USED('clienti')
  USE clienti IN 0 SHARED
ENDIF

SELECT CLIENTI
LOCATE FOR CodCl = 1003
nr_ = 1
FOR i = 1 TO 15000
  succes_ = .F.
```

```
DO WHILE !succes_
    sir1_ = " " + ALLT(STR(nr_))
    IF RLOCK()
        WAIT "Modific inregistrarea... " WINDOW NOWAIT
        REPLACE Adresa WITH "Prosperitatii, "+sir1_
        UNLOCK
        WAIT "Am modificat inregistrarea. SIR1_ este " + sir1_    ;
                WINDOW NOWAIT TIMEOUT 0.01
        nr_ = nr_ + 1
        succes_ = .T.
    ELSE
        WAIT  "Inregistrarea e blocata de alta statie" WINDOW NOCLEAR NOWAIT TIMEOUT
.01
    ENDIF
  ENDDO
ENDFOR
RETURN
```

13.3. Despre buffering

Discuţia despre modalităţile de blocare capătă cu totul alte nuanţe atunci când intră în acţiune o facilitate proprie versiunilor „Visual" ale FoxPro – *buffering*-ul. Spre deosebire de modul „clasic" de lucru, prin buffering scrierea pe disc a modificărilor poate fi amânată, astfel încât timpul de blocare este redus la minimum. Pe de altă parte, prin buffering există şi posibilitatea tratării cazurilor în care două staţii modifică aceleaşi valori.

Ideea buferring-ului este următoarea: în momentul editării unei înregistrări (momentul primei modificări), în memoria locală se copiază o instanţă a înregistrării înainte de modificare. În tot acest timp, oricare alt utilizator (staţie) poate accesa şi edita această înregistrare. Atunci când în sesiunea curentă pointerul tabelei este mutat pe o altă înregistrare sau se încearcă propagarea modificărilor curente pe disc, VFP încearcă să blocheze înregistrarea şi verifică dacă alt utilizator a efectuat modificări în linia curentă, comparându-se valorile curente de pe disc cu valorile-martor din memoria locală. Dacă cele două instanţe – cea veche, stocată în memoria locală şi cea nouă de pe disc (copia principală a bazei de date) – sunt identice, atunci modificările sunt „comise" pe baza de date principală; în caz contrar, se declanşează o eroare.

Este important de ştiut că buffering-ul modifică momentul declanşării regulilor de validare. Cele de la nivel de atribut se verifică tot după modificarea valorii atributului, la ieşirea din câmp, însă cele de la nivel de înregistrare, în momentul „comiterii" pe disc. De exemplu, în lipsa buffering-ului, o comandă `APPEND BLANK` în care cheia primară preia o valoare implicită deja existentă declanşează imediat eroarea de unicitate a indexului, iar adăugarea nu mai are loc. Cu buffering însă, înregistrarea se adaugă, iar verificarea unicităţii se face în momentul „scrierii" înregistrării pe disc.

Acest lucru este deosebit de important. Folosind buffering-ul, problema valorilor implicite, care nu trebuie să întoarcă valori care să pericliteze unicitatea cheilor primare, îşi pierde importanţa. De aici posibilitatea eliminării unor proceduri ce pot micşora serios viteza aplicaţiei.

Buffering-ul poate fi stabilit la nivel de *înregistrare* sau la nivel de *tabelă*. În primul caz, în memorie se încarcă numai câte o linie a tabelei editate; la deplasarea pointerului pe o altă

înregistrare sau prin comanda `TABLEUPDATE()`, se încearcă a se scrie conţinutul bufferului. Folosind *table buffering* în buffer sunt încărcate mai multe înregistrări simultan; editarea lor se face în bufferul local, scrierea efectivă pe disc realizându-se prin intermediul funcţiei `TABLEUPDATE()`.

Ambele tipuri de buffering pot bloca înregistrarea sau grupurile de înregistrări în două moduri: *optimistic* şi *pessimistic*. Buffering-ul pesimist este mai sigur, însă blochează cel mai sever înregistrarea/grupul de înregistrări: din momentul încărcării în bufferul local până în momentul scrierii acestuia pe disc, nici un alt utilizator nu poate edita respectiva înregistrare sau grup de înregistrări. Este similar modului „clasic" de blocare din versiunile FoxPro 2.x. În modul *optimistic*, după momentul încărcării în bufferul local, înregistrarea/grupul de înregistrări poate fi editată/editate şi de alţi utilizatori. Verificarea concordanţei valorilor actuale ale înregistrării cu valorile din momentul încărcării în buffer se realizează în momentul scrierii bufferului pe disc; dacă nu există nici o modificare, atunci se face transferul din buffer pe disc; altminteri, se afişează mesajul *Update conflict* (funcţia `TABLEUPDATE()` întoarce valoarea logică `false (.F.)`). Este adevărat, prin funcţia `TABLEUPDATE()` se poate suprascrie valoarea curentă (tot ce au modificat ceilalţi utilizatori se pierde iremediabil) a înregistrării.

Declararea tipului de buffering se realizează prin comanda-funcţie `CURSORSETPROP()`. Astfel, `CURSORSETPROP('Buffering',1)` dezactivează utilizarea bufferului, `CURSORSETPROP('Buffering',2)` stabileşte un buffering de tip *pessimistic* la nivel de înregistrare, `CURSORSETPROP('Buffering',3)` – buffering de tip *optimistic* la nivel de înregistrare, `CURSORSETPROP ('Buffering', 4)` – stabileşte un buffering de tip *pessimistic* la nivel de tabelă, iar `CURSORSETPROP ('Buffering',5)` stabileşte un buffering de tip *optimistic* la nivel de tabelă. `CURSORSETPROP` întoarce valoarea logică `true (.T.)`, dacă operaţiunea a reuşit, şi `false (.F.)` în caz contrar.

Pentru stabilirea buffering-ului (opţiunile 2, 3, 4 şi 5) este necesar ca parametrul `MULTILOCKS` să fie setat pe `ON` (prin comanda `SET MULTILOCKS ON`).

Pentru optimizarea traficului în reţea, o importanţă deosebită o prezintă buffering-ul *optimistic* – de tip 3 şi 5. Este şi motivul pentru care vom limita discuţia următoare numai la acestea.

13.3.1. *Row Optimistic Buffering*

Bufferul este limitat la o singură înregistare. Înregistrarea curentă se încarcă în buffer, pe disc fiind disponibilă pentru editare şi altor utilizatori. Toate modificările locale se fac exclusiv în buffer. Abia atunci când cursorul este deplasat pe altă înregistrare sau se lansează funcţia `TABLEUPDATE()`, sistemul încearcă scrierea pe disc a bufferului. Dacă înregistrarea din buffer a fost modificată între timp de un alt utilizator, VFP generează eroarea `1585` (*Update conflict*), respectiv funcţia `TABLEUPDATE()` întoarce valoarea logică `False`. Schema este prezentată în figura 13.2.

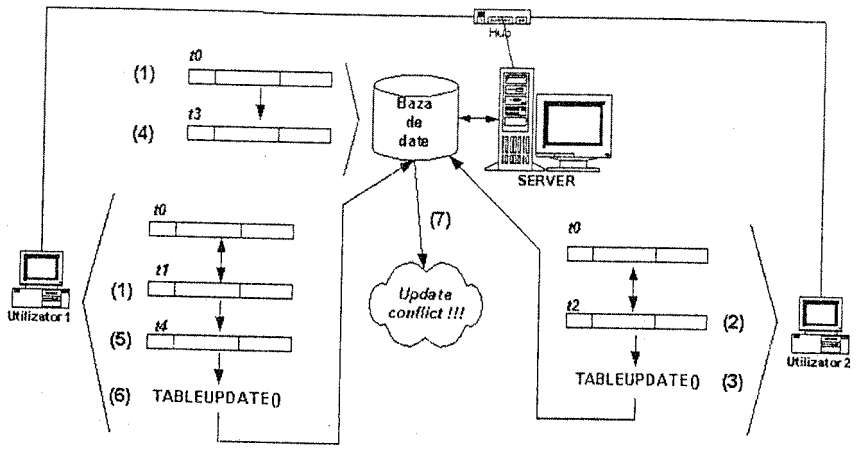


Figura 13.2. Schema buffering-ului la nivel de înregistrare

Firește că în aplicații, pentru controlul actualizărilor, funcția TABLEUPDATE() este foarte utilă. Cei mai importanți parametri sunt:

```
TABLEUPDATE([nRows [, lForce]] [, cTableAlias | nWorkArea] [,
cErrorArray])
```

Argumentul `nRows` este de tip numeric (conform documentației, poate avea una din valorile 0, 1 și 2) și indică, în modul *table optmistic buffering*, dacă se încearcă scrierea tuturor înregistrărilor modificate în buffer sau numai a înregistrării curente. Parametrul `lForce` determină suprascrierea sau nu pe disc a înregistrării cu valorile curente din buffer. Dacă acesta este setat pe `true (.T.)`, atunci valorile din buffer sunt scrise pe disc fără a se mai verifica dacă, între timp, alt utilizator a modificat sau nu valorile originale (din momentul încărcării în buffer) ale înregistrării. Dacă `lForce` este setat pe `false (.F.)` – valoare implicită –, atunci când un alt utilizator a modificat înregistrarea aflată în bufferul local, VFP generează o eroare, iar funcția întoarce `false (.F.)`. Pentru controlul erorilor generate de VFP, se poate folosi o procedură desemnată prin comanda ON ERROR sau, mai elegant, funcția AERROR.

Interesant este și faptul că TABLEUPDATE( ) este operabilă nu numai asupra tabelelor pentru care s-a stabilit o formă de buffering, ci și asupra celor care prezintă reguli de validare de nivel de câmp, înregistrare sau tabelă.

13.3.2. *Table Optimistic Buffering*

După cum îi spune și numele, buffering-ul optimist la nivel de tabelă presupune încărcarea simultană în bufferul local a tuturor înregistrărilor modificate. Înregistrările editate sunt scrise pe disc numai la folosirea explicită a comenzii TABLEUPDATE, prin următoarea secvență:

- se încearcă blocarea fiecărei înregistrări editate;

- după blocare se compară valoarea curentă (de pe server) a fiecărui atribut cu valoarea „originală" (dinaintea modificărilor locale);
- dacă nu sunt diferențe între valorile curente și cele originale, se scrie înregistrarea pe disc (server);
- dacă valorile diferă, se generează o eroare.

Întrucât o eroare poate să apară la a *n*-a înregistrare dintre cele modificate, este recomandabil ca TABLEUPDATE să apară într-o tranzacție (acest lucru nu era atât de important în cazul buffering-ului la nivel de înregistrare).

13.4. Formular în care se folosește buffering optimist la nivel de înregistrare

Pentru a ilustra practic modul de folosire a buffering-ului optimist la nivel de linie, pornim de la un formular simplu ce vizează actualizarea tabelei PRODUSE. Un plus de simplitate a fost obținut prin eliminarea (pe nedrept) a atributului `Imagine`, care e de tip *General*. Formularul se numește `produs0` și e prezentat în figura 13.3.

Cele patru butoane navigaționale din partea superioară sunt denumite `cmdPrimul`, `cmdPrecedentul`, `cmdUrmatorul` și `cmdUltimul`, cele trei de editare – `cmdProdusNou`, `cmdModificare` și `cmdStergere`, iar cele din partea inferioară sunt `cmdOK` și `cmdAbandon`. Căsuțele de text au același nume ca al atributului la care sunt legate (prin proprietatea `ControlSource`), prefixate, după notația ungurească, cu `txt`.

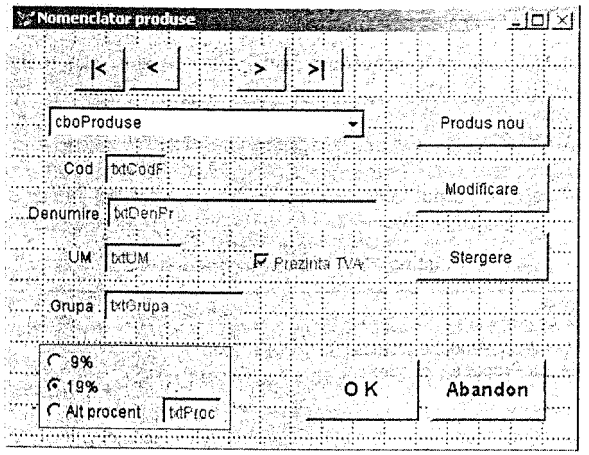


Figura 13.3. Formularul PRODUSE0

Căsuța de validare a fost botezată `chkPrezintaTVA`, iar grupul de butoane de opțiune, `optTVA`. În privința combo-box-ului, cele mai importante proprietăți se prezintă astfel: *RowSourceType*: 3 – SQL Statement, iar *RowSource*: `SELECT DenPr, CodPr FROM produse INTO CURSOR cProduse ORDER BY DenPr`. Formularul mai prezintă și două proprietăți-utilizator, `nivel_tranzactie` și `succes` folosite pe parcursul metodelor ce urmează a fi descrise în continuare.

Începem prezentarea metodelor cu `INIT` din listingul 13.6. Se verifică dacă baza de date este deschisă şi, în caz contrar, se deschide în mod partajat şi se lansează procedura stocată pentru deschiderea tuturor tabelelor – `setup`. Prin `CURSORSETPROP` se declară buffering-ul de tip optimist la nivel de înregistrare.

Listing 13.6. Metoda `Init` a formularului

```
* se deschide baza de date (daca nu e deja deschisa)
IF !DBUSED('vinzari')
  OPEN DATA vinzari SHARED
  DO setup
ENDIF

SELECT produse
* se dezactiveaza Buffering-ul
=CURSORSETPROP('Buffering', 3)

* se stabileste indexul principal al tabelei
SET ORDER TO TAG CodPr

* se rezolva problema inregistrarii curente
IF EOF() OR BOF()
  GO BOTTOM
ENDIF

THISFORM.Nivel_Tranzactie = TXNLEVEL()

* se apeleaza metoda de reimprospatare a formularului
THISFORM.Refreseaza
```

În plus, nu trebuie uitat ca în procedura principală – `setup` – să mai fie adăugate (cel puţin) două comenzi:

```
SET DELETE ON
SET MULTILOCKS ON
```

Listingul 13.7 conţine corpul metodei `refreseaza`, necesară în completarea metodei-sistem `refresh` pentru sincronizarea combo-box-ului şi grupului de butoane de opţiune.

Listing 13.7. Metoda `refreseaza`

```
THISFORM.Refresh
FOR i = 1 TO THISFORM.cboProduse.ListCount
  IF VAL(THISFORM.cboProduse.List(i,2)) = Produse.CodPr
      THISFORM.cboProduse.ListIndex = i
      EXIT
  ENDIF
ENDFOR

DO CASE
CASE EMPTY(produse.procTVA)
  THISFORM.chkPrezintaTVA.Value = 0
  THISFORM.optTVA.Value = 3
CASE produse.procTVA = .09
  THISFORM.chkPrezintaTVA.Value = 1
  THISFORM.optTVA.Value = 1
CASE produse.procTVA = .19
  THISFORM.chkPrezintaTVA.Value = 1
  THISFORM.optTVA.Value = 2
OTHERWISE
  THISFORM.chkPrezintaTVA.Value = 1
  THISFORM.optTVA.Value = 3
ENDCASE
THISFORM.txtProcTVA.Value = produse.procTVA
```


Pentru activarea căsuţelor de tip text (simultan cu dezactivarea celorlalte controale – combo-box-ul, butoanele de navigare şi cele de adăugare/modificare/ştergere) se foloseşte metoda `activare_controale` (listingul 13.8).

Listing 13.8. Metoda `activare_controale`

```
WITH THISFORM
  .cmdPrimul.Enabled=.F.
  .cmdPrecedentul.Enabled=.F.
  .cmdUrmatorul.Enabled=.F.
  .cmdUltimul.Enabled=.F.
  .cboProduse.Enabled=.F.
  .cmdProdusNou.Enabled=.F.
  .cmdModificare.Enabled=.F.
  .cmdStergere.Enabled=.F.
  .txtDenPr.Enabled=.T.
  .txtUM.Enabled=.T.
  .txtGrupa.Enabled=.T.
  .chkPrezintaTVA.Enabled=.T.
ENDWITH

IF THISFORM.chkPrezintaTVA.Value = 1
  THISFORM.optTVA.Enabled=.T.
ELSE
  THISFORM.optTVA.Enabled=.F.
ENDIF
```

După editarea unei înregistrări este necesară confirmarea sau abandonarea modificărilor, iar apoi activarea butoanelor de navigare şi „îngheţarea" căsuţelor de tip text, astfel încât este necesară o metodă specială numită `dezactivare_controale` (vezi listingul 13.9).

Listing 13.9. Metoda `dezactivare_controale`

```
WITH THISFORM
  .cmdPrimul.Enabled=.T.
  .cmdPrecedentul.Enabled=.T.
  .cmdUrmatorul.Enabled=.T.
  .cmdUltimul.Enabled=.T.
  .cboProduse.Enabled=.T.
  .cmdProdusNou.Enabled=.T.
  .cmdModificare.Enabled=.T.
  .cmdStergere.Enabled=.T.
  .txtDenPr.Enabled=.F.
  .txtUM.Enabled=.F.
  .txtGrupa.Enabled=.F.
  .chkPrezintaTVA.Enabled=.F.
  .optTVA.Enabled=.F.
ENDWITH
```

Trecem la descrierea principalelor metode pentru controalele formularului. La selectarea unei opțiuni din combo-box se mută pointerul pe înregistrarea corespunzătoare din cursorul `cProduse` (datorită proprietăților `RowSourceType` și `RowSource` ale combo-ului). Metoda `InteractiveChange` poziționează cursorul în tabela `PRODUSE` pe înregistrarea aferentă din cursor `cProduse` – vezi listingul 13.10.

Listing 13.10. Metoda `InteractiveChange` pentru controlul `cboProduse`

```
=SEEK (cProduse.CodPr, 'produse', 'codpr')
THISFORM.Refreseaza
```

Pentru butoanele de navigare, singura metodă care interesează este `Click`, care este asemătoare – listingurile 13.11, 13.12, 13.13 și 13.14.

Listing 13.11. Metoda asociată evenimentului `Click` pentru controlul `cmdPrimul`

```
SELECT cProduse
GO TOP
THISFORM.cboProduse.InteractiveChange
```

Listing 13.12. Metoda asociată evenimentului `Click` pentru controlul `cmdPrecedentul`

```
SELECT cProduse
IF !BOF()
  SKIP -1
ELSE
  GO TOP
ENDIF
THISFORM.cboProduse.InteractiveChange
```

Listing 13.13. Metoda asociată evenimentului `Click` pentru controlul `cmdUrmatorul`

```
SELECT cProduse
IF !EOF()
  SKIP
  IF EOF()
     GO BOTTOM
  ENDIF
ELSE
  GO BOTTOM
ENDIF
THISFORM.cboProduse.InteractiveChange
```

Listing 13.14. Metoda asociată evenimentului `Click` pentru controlul `cmdUltimul`

```
SELECT cProduse
GO BOTTOM
THISFORM.cboProduse.InteractiveChange
```

Dintre căsuțele de tip text, doar `txtProcTVA` prezintă o metodă prin care se actualizează valoarea atributului `PRODUSE.ProcTVA` cu valoarea acestui control – vezi listingul 13.15.

Listing 13.15. Metoda `Valid` pentru controlul `txtProcTVA`

```
REPLACE produse.procTVA WITH THISFORM.txtProcTVA.Value
```

Metoda `Click` a căsuței de validare `chkPrezintaTVA` este cea din listingul 13.16, iar metoda asociată aceluiași eveniment pentru grupul butoanelor de opțiune este prezentată în listingul 13.17.

Listing 13.16. Metoda `Click` pentru controlul `chkPrezintaTVA`

```
IF THIS.Value = 1   && Inainte produsul nu prezenta TVA, acum DA !
  THISFORM.optTVA.Enabled = .T.
  THISFORM.optTVA.Value = 2
  THISFORM.txtProcTVA.Value = 19
ELSE      && Inainte prezenta TVA, acum NU
  THISFORM.optTVA.Value = 3
  THISFORM.txtProcTVA.Value = 0
  THISFORM.optTVA.Enabled = .F.
ENDIF
THISFORM.txtProcTVA.Valid
```

Listing 13.17. Metoda `Click` pentru grupul butoanelor de opțiune (`optTVA`)

```
DO CASE
CASE THIS.Value = 1        && primul buton (9%) este marcat
  THISFORM.txtProcTVA.Value = .09
  THISFORM.txtProcTVA.Enabled = .F.
CASE THIS.Value = 2                && al doilea buton (19%) este marcat
  THISFORM.txtProcTVA.Value = .19
  THISFORM.txtProcTVA.Enabled = .F.
OTHERWISE                  && al treilea buton din grup (alt procent) este marcat
  THISFORM.txtProcTVA.Enabled = .T.
  THISFORM.txtProcTVA.SetFocus
ENDCASE
THISFORM.txtProcTVA.Valid
```

Ne apropiem de miezul evenimentelor. Listingul 13.18 conține corpul metodei asociate evenimentului `Click` pentru controlul `cmdProdusNou`. Ideea este ca, la adăugare, să se recupereze o eventuală înregistrare marcată pentru ștergere. Artificiul se poate dovedi foarte util, câtă vreme singurul mod de a elimina fizic înregistrările marcate pentru ștergere ține de folosirea comenzii `PACK`, imposibil de lansat în modul `SHARED`.

Listing 13.18. Metoda `Click` pentru `cmdProdusNou`

```
SELECT produse
SET DELE OFF
LOCATE FOR DELETED()
IF EOF()
  APPEND BLANK
ELSE
  RECALL
  REPLACE CodPr with def_codpr_produse()
ENDIF
REPLACE DenPr with "<Produs Nou>", UM WITH 'buc', grupa WITH " ", ProcTVA WITH .19

SET DELETE ON
```

```
THISFORM.Refreseaza
THISFORM.Activare_Controale
THISFORM.txtDenPr.SetFocus
```

Metoda asociată evenimentului `Click` pentru butonul de control `cmdModificare` (listingul 13.19), prin care se poate trece la modificarea datelor generale despre produse, este una dintre cele mai simple.

Listing 13.19. Metoda `Click` pentru `cmdModificare`

```
THISFORM.Activare_Controale
THISFORM.txtDenPr.SetFocus
```

Aceeaşi metodă a controlului `cmdStergere` (listingul 13.20) trebuie să dea utilizatorului posibilitatea să evite ştergerile accidentale, scop pentru care se afişează un mesaj care cere confirmarea ştergerii.

Listing 13.20. Metoda `Click` pentru `cmdStergere`

```
IF MESSAGEBOX('Chiar trebuie sters produsul '+ALLT(produse.denpr)+' ?',4+32+0) = 6 && Yes
  SET DELETE OFF
  SELECT produse
  BEGIN TRAN
  ON ERROR A=1
  DELETE
  IF DELETED()
      END TRAN
      IF !EOF()
          SKIP
      ELSE
          SKIP -1
      ENDIF
      THISFORM.cboProduse.Requery
      THISFORM.Refreseaza
  ELSE
      ROLLBACK
  ENDIF
ENDIF
ON ERROR
SET DELETE ON
```

Au mai rămas în discuţie metodele asociate evenimentului `Click` ale celor două butoane de comandă ce sunt disponibile pe tot parcursul sesiunii de lucru (cu formularul), `cmdOK` şi `cmdAbandon`. Începem cu al doilea, deoarece lucrurile sunt mai simple în acest caz – listingul 13.21.

Listing 13.21. Metoda `Click` pentru `cmdAbandon`

```
SELECT produse
IF '2' $ GETFLDSTATE(-1) OR '3' $ GETFLDSTATE(-1) OR '4' $ GETFLDSTATE(-1)
  IF THISFORM.Nivel_Tranzactie < TXNLEVEL()
      ROLLBACK
  ENDIF
  =TABLEREVERT()
ELSE
  IF MESSAGEBOX('Doriti sa inchideti formularul ?',4+32+0) = 6 && Yes
      DO WHILE THISFORM.Nivel_Tranzactie < TXNLEVEL()
```



```
          ROLLBACK
      ENDDO
      THISFORM.Release
  ENDIF
ENDIF
THISFORM.Refreseaza
THISFORM.Dezactivare_Controale
```

Această metodă este prima ce utilizează funcţii şi opţiuni specifice buffering-ului. Astfel, pe a doua linie, prin funcţia `GETFLDSTATE` se testează dacă înregistrarea curentă este una nouă sau dacă au fost operate modificări în înregistrarea curentă. Argumentul –1 al funcţiei `GETFLDSTATE` furnizează un şir de caractere ce indică atât starea fiecărui câmp în parte, cât şi starea de ansamblu a înregistrării.

Dacă au fost operate inserări sau modificări, se închid eventualele tranzacţii deschise de la intrarea în formular (scop în care este folosită proprietatea `Nivel_Tranzactie` a formularului) şi se „curăţă" bufferul (implicit, se anulează toate modificările din buffer). Acţionarea acestui buton în condiţiile inexistenţei vreunei modificări constituie semnalul că utilizatorul vrea să închidă formularul, astfel încât i se cere confirmarea închiderii.

Este de presupus, totuşi, ca utilizatorul să mai şi salveze ceea ce a lucrat, aşa încât cel mai important buton al formularului, cel prin acţionarea căruia se „consfinţesc" modificările, este `cmdOK`. Metoda asociată acestui eveniment este cea din listingul 13.22.

Listing 13.22. Metoda `Click` pentru `cmdOK`

```
SELECT produse
IF '2' $ GETFLDSTATE(-1) OR '3' $ GETFLDSTATE(-1) OR '4' $ GETFLDSTATE(-1)
  THISFORM.ScrieBuffer
  IF THISFORM.Succes
      THISFORM.cboProduse.Requery
      THISFORM.Refreseaza
      THISFORM.Dezactivare_Controale
  ELSE
      THISFORM.txtDenPr.SetFocus
  ENDIF
ELSE
  IF MESSAGEBOX('Doriti sa inchideti formularul ?',4+32+0) = 6 && Yes
      DO WHILE THISFORM.Nivel_Tranzactie < TXNLEVEL()
          END TRANSACTION
      ENDDO
      THISFORM.Release
  ENDIF
ENDIF
```

Ca şi pentru butonul `cmdAbandon`, se verifică dacă au fost operate modificări în tabela `PRODUSE`. Dacă dă, se apelează o metodă specială pentru „scrierea" bufferului pe disc (`ScrieBuffer`) şi, la revenirea din această metodă, se verifică dacă „scrierea" a fost un succes (caz în care se dezactivează text-box-urile şi se activează butoanele navigaţionale şi combo-ul) sau un eşec, situaţie în care se continuă cu editarea înregistrării curente, predându-se controlul către `txtDenPr`.

Prin urmare, metoda-cheie în scrierea bufferului pe disc şi tratarea eventualelor conflicte şi altor erori este `ScriereBuffer`, prezentată în listingul 13.23.

Listing 13.23. Metoda `ScrieBuffer`

```
SELECT produse
SET REPROCESS TO 1
deschis_ = .f.
WAIT 'Asteptati pina se deblocheaza inregistrarea' WINDOW AT 30,30 NOWAIT NOCLEAR
DO WHILE !deschis_
  IF RLOCK()
     deschis_ = .t.
  ELSE
     WAIT 'Asteptati pina se deblocheaza inregistrarea !' window at 30,30 nowait noclear
  ENDIF
ENDDO
WAIT CLEAR
ON ERROR
PrimaOara = .T.
gata_ = .F.

BEGIN TRANSACTION
DO WHILE !gata_
  IF PrimaOara
     THISFORM.Succes = TABLEUPDATE(1, .F.)
     PrimaOara = .F.
  ELSE
     THISFORM.Succes = TABLEUPDATE(1, .T.)
     IF !THISFORM.Succes
        IF MESSAGEBOX('Abandonati modificarile de pe linia curenta ?',4+32+0) = 6 && Yes
              ROLLBACK
              TABLEREVERT()
              gata_ = .T.
        ENDIF
     ENDIF
  ENDIF

  IF !THISFORM.Succes
     AERROR(erori)

     DO CASE
        CASE erori[1,1] = 1884                  && unicitate violata
              IF erori[1,3] = 'CODPR'           && de la cheia primara ni se trage
                    REPLACE codpr WITH def_codpr_produse()
                    PrimaOara = .T.
              ELSE
                    && se dubleaza alte chei candidat
                    PrimaOara = .T.
                    MESSAGEBOX('Valoarea aferenta indexului '+erori(1,3)+' se repeta !!!')
           ROLLBACK
           RETURN
              ENDIF

        CASE erori[1,1] = 1585 OR erori[1,1] = 1595
           FOR i = 1 TO FCOUNT() - 1 && ultimul cimp este de tip General
                 modificat_ = .t.
                 cimp_ = FIELD(i)
                 IF OLDVAL(cimp_) <> CURVAL(cimp_)
                       CeFac_ = MESSAGEBOX('Si alt utilizator a modificat - '+;
                             cimp_+','+CHR(13)+' Le pastram pe ale noastre ? ',;
                             4+48)
                       IF CeFac_ = 7
                             REPLACE (cimp_) WITH CURVAL(cimp_)
                       ENDIF
                 ENDIF
           ENDFOR

     ENDCASE
  ELSE
     END TRANSACTION
     gata_ = .T.
  ENDIF
ENDDO
UNLOCK
```


Deşi prin folosirea tranzacţiilor înregistrarea se blochează automat, pentru a semnaliza situaţiile de blocaj se foloseşte sistemul „manual" expus în paragrafele anterioare.

Prima încercare de scriere a bufferului pe disc este fără forţarea suprascrierii (al doilea argument al funcţiei `TABLEUPDATE` este .F.). În caz că apare vreo eroare, informaţiile despre aceasta sunt preluate prin funcţia `AERROR` în vectorul `erori`.

Una dintre situaţiile-problemă tratate este cea în care doi sau mai mulţi utilizatori adaugă simultan câte un produs. Funcţia stocată `def_codpr_produse` întoarce aceeaşi valoare a `PRODUSE.CodPr`, iar la comiterea adăugărilor pe disc se declanşează eroarea `1884` datorată duplicării valorii unei chei primare (sau candidat). Problema se rezolvă elegant prin apelarea repetată a funcţiei care întoarce valoarea implicită până când restricţia nu mai este încălcată.

Când apare o eroare tot cu numărul `1884`, dar sursa este indexul candidat `DenPr`, se returnează controlul metodei apelante – metoda evenimentului `Click` al controlului `cmdOK` –, metodă care continuă editarea înregistrării curente.

Atunci când doi utilizatori modifică simultan aceeaşi înregistrare, după ce primul salvează pe disc conţinutul bufferului, în momentul scrierii bufferului celui de-al doilea utilizator se declanşează eroarea `1585`. Metoda propusă trece în revistă toate atributele înregistrării curente (cu ajutorul funcţiilor `FCOUNT` şi `FIELD`), comparând valorile dinaintea editărilor (`OLDVAL`) cu cele actuale din baza de date „master", adică de pe server (`CURVAL`). Când cele două valori diferă, se afişează ceea ce a fost introdus de alt utilizator, iar utilizatorul curent este întrebat dacă se păstrează valoarea de pe server sau valoarea din bufferul local.

După trecerea prin toate atributele, variabila `PrimaOara` ia valoarea .F. şi se revine la începutul buclei, încercându-se scrierea forţată a bufferului. Dacă şi acum scrierea nu e reuşită (continuă să se violeze o restricţie de cheie sau o regulă la nivel de înregistrare sau un declanşator), utilizatorul poate opta între a continua editarea înregistrării curente sau abandona modificările efectuate.

Acest stil de tratare a erorilor în mediile multiutilizator poate fi completat cu alte opţiuni şi alte tipuri de erori ce pot fi depistate şi tratate.

13.5. Formular în care se foloseşte buffering optimist la nivel de tabelă

Pentru a asigura continuitatea atât cu capitolul dedicat procedurilor stocate, cât şi cu problematica formularelor, în cele ce urmează trecem în reţea formularul `Produse` (`frmProduse`) prezentat în paragraful 10.1 – vezi figura 13.4.

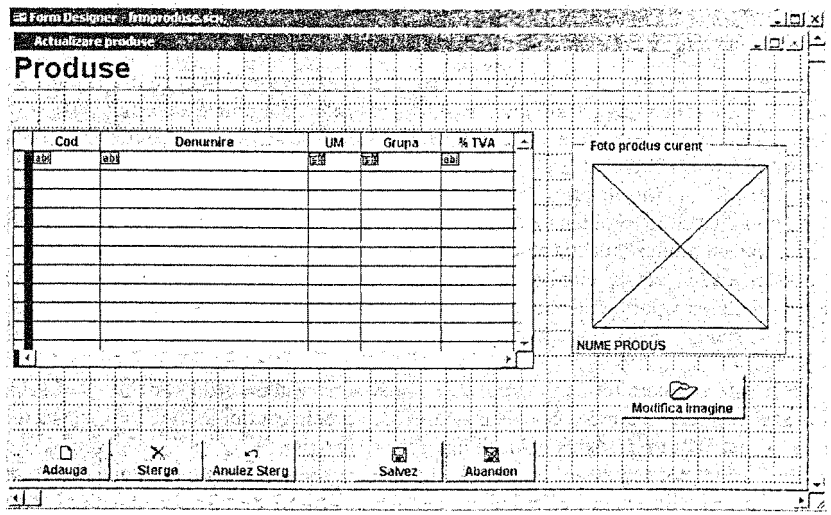


Figura 13.4. Formularul `frmProduse` pentru care se utilizează *Table Optimistic Buffering*

Principala problemă a acestui formular este legat de controlul de tip reţea (grid) care, pe de o parte, face procedura de editare a datelor din `PRODUSE` mult mai lejeră, în comparaţie cu formularul din paragraful precedent, iar pe de altă parte, face greu de controlat programatic momentul de scrierere a bufferului pe disc. Aşa încât soluţia buffering-ului la nivel de tabelă a fost cea mai la îndemână pentru a-l „porta" pentru lucrul în reţea.

Păstrăm proprietăţile şi mare parte din metodele prezentate în capitolul 10, insistând cu precădere asupra diferenţelor. Introducem următoarele proprietăţi la nivel de formular: `Adresa` – pentru a memora linia la care se va reveni după scrierea tuturor înregistrărilor din buffer, `NivelTranzactie` – pentru „păstrarea" nivelului tranzacţiilor la intrarea în formular, astfel încât la ieşire să se respecte acest nivel, `InregistrareProcesata` – utilă în scrierea fiecărei înregistrări din buffer (şi tratarea eventualelor erori), `Succes` – dacă scrierea unei linii din buffer a decurs fără sau cu probleme.

Metoda-cheie a formularului pentru funcţionarea în reţea este, ca şi în paragraful precedent, `ScriereBuffer`, pe care o vom discuta în final.

Începem prezentările cu „noul look" al metodei `Load` – listingul 13.24 –, care verifică dacă baza de date şi tabela `PRODUSE` sunt deschise şi, în afara iniţializării proprietăţilor `NivelTranzacţie` şi `Succes`, stabileşte tipul de buffering.

Listing 13.24. Metoda `Load` a formularului `frmProduse`

```
IF !DBUSED('vinzari')
  OPEN DATABASE vinzari SHARED
  DO setup
ELSE
  IF !USED('produse')
      USE produse IN 0 SHARED
  ENDIF
ENDIF

* se verifica daca tabelele sunt deschise
* (sau daca pot fi deschise) in mod SHARED
SELECT produse
CURSORSETPROP('BUFFERING',5)

* se memoreaza nivelul tranzactiei la intrarea in formular
* NivelTranzactie este o proprietate a formularului
THISFORM.NivelTranzactie = TXNLEVEL()
THISFORM.Succes = .T.
```

Având în vedere modul de derulare a operaţiunilor în acest tip de buffering, controlul de tip `Grid`, precum şi obiectele sale nu (mai) prezintă metode-utilizator. Imediat după lansarea formularului, utilizatorul poate intra în grid şi edita oricâte produse. Abia la „acţionarea" butonului `Save` intră în operă tranzacţia de scriere pe disc a fiecărei linii din buffer. Iată, în listingul 13.25, conţinutul metodei asociate evenimentului `Click` pentru butonul de comandă `cmdSave`.

Listing 13.25. Metoda asociată evenimentului `Click` pentru `cmdSave`

```
SELECT produse
IF !THISFORM.Succes
  GO MIN(THISFORM.InregistrareProcesata, RECCOUNT())
  THISFORM.grdProduse.columnCodPr.Text1.SetFocus
ELSE
  IF GETNEXTMODIFIED(0) <> 0
      THISFORM.ScriereBuffer
      IF !THISFORM.Succes
          GO MIN(THISFORM.InregistrareProcesata, RECCOUNT())
          THISFORM.grdProduse.columnCodPr.Text1.SetFocus
      ELSE
          IF THISFORM.NivelTranzactie < TXNLEVEL()
                END TRANSACTION
          ENDIF
      ENDIF
  ELSE
      IF MESSAGEBOX('Doriti sa inchideti formularul ?',4+32+0) = 6 && Yes
          SET DELE ON
          THISFORM.Release
      ENDIF
  ENDIF
ENDIF
```

Primul `IF` verifică dacă acţionarea butonului `Save` s-a făcut fără a trata eventualele erori apărute la scrierea unei înregistrări precedente. După cum vom vedea peste câteva rânduri, în metoda `ScriereBuffer`, dacă o înregistrare generează probleme la scrierea pe disc (server), proprietatea `Succes` a formularului are valoarea `.F.`. Astfel încât acest prim `IF` trimite utilizatorul în grid (pentru a rezolva eroarea), oricât de insistent ar apăsa butonul `Save`.

Dacă nu există nici o problemă anterioară nerezolvată (ramura ELSE a IF-ului principal), se verifică, prin funcția GETNEXTMODIFIED, dacă a fost operată măcar o modificare, caz în care se apelează metoda ScriereBuffer. La ieşirea din ScriereBuffer se testează dacă au apărut probleme şi, dacă da, se cedează controlul grid-ului pentru rezolvarea erorii. Dacă nu sunt erori, se încheie tranzacția (deschisă tot în ScriereBuffer).

Ultimul IF este operațional atunci când utilizatorul acționează butonul Save fără a fi făcut nici o adăugare, modificare sau ştergere, ceea ce ar fi un indiciu că se doreşte ieşirea din formular, astfel încât se cere confirmarea ieşirii.

A sosit momentul să discutăm câte ceva şi despre mult invocata (apelata) metodă ScriereBuffer. Iat-o în listingul 13.26.

Listing 13.26. Metoda ScriereBuffer

```
SELECT produse
THISFORM.adresa = MAX(RECNO(),RECCOUNT())

*  prima inregistrare editata
THISFORM.InregistrareProcesata = GETNEXTMODIFIED(0)
IF THISFORM.InregistrareProcesata <> 0
  * daca e cel putin o inregistrare editata,
  * se marcheaza inceputul unei tranzactii
  BEGIN TRANSACTION
ENDIF

gata_toate_inregistrarile = .F.

DO WHILE THISFORM.InregistrareProcesata <> 0  AND ;
      !gata_toate_inregistrarile                  && bucla principala
  * pozitionarea pointerului pe inregistrarea
  * ce urmeaza a fi scrisa din buffer
  GO THISFORM.InregistrareProcesata

  gata_inregistrare = .F.
  PrimaOara = .T.
  DO WHILE !gata_inregistrare                     && bucla inregistrarii curente
      IF PrimaOara
          * prima data, se incearca "comiterea" inregistrarii fara
          * a forta suprascrierea
          THISFORM.Succes = TABLEUPDATE(0, .F.)
          PrimaOara = .F.
      ELSE
          * dupa alegerea valorilor ce erau in conflict,
          *       inregistrarea se suprascrie
          THISFORM.Succes = TABLEUPDATE(0, .T.)
          IF !THISFORM.Succes
                TABLEREVERT(0)
                gata_inregistrare = .T.
          ENDIF
      ENDIF
      IF !THISFORM.Succes AND !gata_inregistrare
          * tratarea erorilor
          AERROR(erori)
          DO CASE
                CASE erori[1,1] = 1884                  && unicitate violata
                      IF erori[1,3] = 'CODPR'           && de la cheia primara ni se trage
```



```
                            REPLACE codpr WITH def_codpr_produse()
                            PrimaOara = .T.     && pentru a nu forta suprascrierea
                      ELSE
                            && se dubleaza alte chei candidat
                            PrimaOara = .T.
                            MESSAGEBOX('Valoarea aferenta indexului '+;
                            erori(1,3)+' se repeta !!!')
                            RETURN
                      ENDIF

                CASE erori[1,1] = 1585 OR erori[1,1] = 1595
                      FOR i = 1 TO FCOUNT()
                            modificat_ = .t.
                            cimp_ = FIELD(i)
                            IF OLDVAL(cimp_) <> CURVAL(cimp_)
                                  IF UPPER(cimp_) = 'CODPR' AND RECNO() < 0
                                  * Problema e la CodPr al unei inregistrari adaugate
                                        REPLACE codpr WITH;
                                              def_codpr_produse()
                                        * pentru a nu forta suprascrierea
                                        PrimaOara = .T.
                                        EXIT
                                  ELSE
                                  * e alt atribut decit CodPr ;
                                  *     sau nu e vorba de adaugare
                                        CeFac_ = MESSAGEBOX(;
                                        'Si alt utilizator a modificat - '+;
                                        cimp_+'. - '+CHR(13)+;
                                        'Pastram valorile noastre ? ',4+48)
                                        IF CeFac_ = 7
                                              REPLACE (cimp_) ;
                                              WITH CURVAL(cimp_)
                                        ENDIF
                                  ENDIF
                            ENDIF
                      ENDFOR

                CASE erori[1,1] = 1539                  && TRIGGER FAILED !
                      * daca e vorba de o stergere a unei inregistrari parinte,
                      *       ii anulam marcajul de stergere
                      IF LEFT(GETFLDSTATE(-1),1) = '2'
                            RECALL
                      ENDIF
                      gata_inregistrare = .T.
                      THISFORM.Succes = .T.
          ENDCASE
      ELSE
          gata_inregistrare = .T.
      ENDIF
  ENDDO
  && urmatoarea inregistrare editata
  THISFORM.InregistrareProcesata = GETNEXTMODIFIED(0)
ENDDO
IF THISFORM.NivelTranzactie < TXNLEVEL()
  END TRANSACTION
ENDIF
GO THISFORM.Adresa
```

Obiectivul declarat al acestei metode este parcurgerea fiecărei înregistrări din bufferul tabelei (înregistrare ce a fost adăugată, modificată sau ştearsă), scrierea sa pe server şi, eventual, rezolvarea diverselor probleme ce pot să apară la scriere. A doua comandă a metodei, `THISFORM.adresa = MAX(RECNO(),RECCOUNT())`, stochează în proprietatea formularului, folosită de noi drept (un soi de) variabilă publică, adresa înregistrării curente *înainte* de începerea ostilităţilor. Aceasta va fi adresa la care VFP se va întoarce după scrierea înregistrărilor. Întrebarea este: de ce s-a folosit funcţia MAX? Ei, bine, orice înregistrare audăugată în buffer are RECNO-ul negativ, iar după „comiterea" pe disc, numărul acesteia va fi atribut normal. Dacă după scriere am folosi o comandă GO şi un argument negativ (adresa înregistrării agăugate înainte de scriere), este evident că nu am putea rata un mesaj de eroare.

Funcţia `GETNEXTMODIFIED` este o facilitate-cheie în lucrul cu buffering la nivel de tabelă, deoarece furnizează adresa următoarei înregistrări editate din buffer. Adresa înregistrării curente (în curs de scriere pe server) este stocată în proprietatea formularului numită `InregistrareProcesata`. Este suficient să existe măcar o înregistrare procesată pentru a se declanşa o tranzacţie.

Bucla principală a metodei are ca scop parcurgerea tuturor înregistrărilor editate. În cadrul acesteia se află o a doua buclă, asemănătoare formularului din paragraful precedent, buclă în care se încearcă scrierea şi rezolvarea eventualelor probleme legate de o înregistrare. Un caz nou tratat este cel în care a fost marcată pentru ştergere o înregistrare-părinte, ştergerea atrăgând după sine încălcarea restricţiei referenţiale.

Această metodă este apelată şi din cele ale butoanelor de adăugare şi ştergere. Astfel, listingul 13.27 conţine metoda asociată evenimentului `Click` pentru butonul de adăugare – `cmdAdauga`.

Listing 13.27. Metoda asociată evenimentului `Click` pentru `cmdAdauga`

```
IF !THISFORM.Succes      && nu s-au rezolvat conflictele sau alte probleme din editari anterioare
  MESSAGEBOX('Rezolvati problemele actuale din buffer si apoi adaugati noi inregistrari')
  THISFORM.grdProduse.columnCodPr.Text1.SetFocus
  RETURN
ENDIF

THISFORM.Succes = .T.
* daca sun modificari in buffer, se comit
THISFORM.ScriereBuffer
SELECT produse
IF !THISFORM.Succes
  GO MIN(THISFORM.InregistrareProcesata, RECCOUNT())
  THISFORM.grdProduse.columnCodPr.Text1.SetFocus
ELSE
  SET DELETE OFF
  LOCATE FOR DELETED()
  IF EOF()
      APPEND BLANK
      REPLACE denpr WITH 'Produs Nou', um WITH 'L', grupa WITH 'Bere', ProcTVA WITH .19
  ELSE
      * reciclam o inregistrare marcata pentru stergere
      RECALL
      REPLACE codpr WITH def_codpr_produse(), denpr WITH 'Produs_nou', ;
          um WITH 'L', grupa WITH 'Bere', ProcTVA WITH .19
  ENDIF
  SET DELETE ON
ENDIF
THISFORM.grdproduse.columnCodpr.SETFOCUS
```


Ca şi în cazul butonului de salvare, primul `IF` obligă utilizatorul să rezolve mai întâi problemele apărute la scrierea înregistrării curente, iar mai apoi să adauge noi înregistrări. Pentru a mai diminua din riscuri, adăugarea înregistrărilor se realizează în regim de pseudo-row buffering, adică, înaintea adăugării unei înregistrări, se verifică dacă bufferul conţine vreo înregistrare editată şi, dacă da, se scrie bufferul pe disc. Adăugarea decurge asemănător formularului din paragraful precedent, „reciclându-se" eventuale înregistrări marcate pentru ştergere.

Metoda butonului de ştergere – listingul 13.28 – tratează diferit cazurile în care se şterge o înregistrare „veche", încărcată de pe server, sau o înregistrare proaspăt adăugată, ce nu a fost scrisă din buffer.

Listing 13.28. Metoda asociată evenimentului `Click` pentru `cmdSterge`

```
IF !THISFORM.Succes      && nu s-au rezolvat conflictele sau alte probleme din editari anterioare
  MESSAGEBOX('Rezolvati problemele actuale din buffer si apoi adaugati noi inregistrari')
  THISFORM.grdProduse.columnCodPr.Text1.SetFocus
  RETURN
ENDIF
THISFORM.Succes = .T.
* daca sun modificari in buffer, se comit
SELECT produse

SET DELE OFF
IF RECNO() < 0   && se doreste stergerea unei inregistrari tocmai adaugate (din buffer)
  =TABLEREVERT(.F.)              &&      se elimina din buffer numai inregistrarea curenta
ELSE
  DELETE
ENDIF
THISFORM.asters=.T.
THISFORM.REFRESH
```

Una dintre cele mai plăcute metode este cea asociată evenimentului `click` pentru butonul de comandă dedicat anulării unei ştergeri – vezi listingul 13.29.

Listing 13.29. Metoda asociată evenimentului `Click` pentru `cmdAnulezSterg`

```
SELECT produse
RECALL
```

Despre metoda din listingul 13.30 n-ar mai fi de spus decât că închide toate eventualele tranzacţii deschise, până la nivelul intrării în formular, şi cere confirmarea utilizatorului pentru închiderea sesiunii.

Listing 13.30. Metoda asociată evenimentului `Click` pentru `cmdAbandon`

```
SELECT produse
DO WHILE THISFORM.NivelTranzactie < TXNLEVEL()
  ROLLBACK
ENDDO
```

```
IF MESSAGEBOX('Doriti sa inchideti formularul ?',4+32+0) = 6 && Yes
  DO WHILE THISFORM.NivelTranzactie < TXNLEVEL()
     ROLLBACK
  ENDDO
  SET DELE ON
  THISFORM.Release
ENDIF
TABLEREVERT(.T.)  && se abandoneaza toate modificarile din buffer
THISFORM.GrdProduse.ColumnCodpr.SETFOCUS
```

Nici metoda `Error` a formularului (listingul 13.31) nu conține elemente spectaculoase, ci doar preluarea explicită a erorii de unicitate și semnalizarea altor tipuri de erori.

Listing 13.31. Metoda evenimentului `Error`

```
LPARAMETERS nError, cMethod, nLine
IF TYPE('erori') = "U"
  PUBLIC ARRAY erori (7)
ENDIF

=AERROR(erori)
IF nError=1884
   MESSAGEBOX('Valoarea aferenta indexului '+erori(3)+' se repeta !!!',16,'Eroare de unicitate')
ELSE
  MESSAGEBOX('Numar:'+STR(nError)+ALLT(erori(2))+', ;
  Metoda:'+cMethod+', Linia :'+STR(nLine) ,16,;
      'Eroare neprevazuta')
ENDIF
THISFORM.grdProduse.columnCodPr.Text1.SETFOCUS
```

Încheiem cu metoda `Destroy` a formularului, pe care, pentru comoditate, am modificat-o astfel:

Listing 13.32. Metoda asociată evenimentului `Destroy`

```
THISFORM.cmdSave.Click
```

Capitolul 14

Aplicații Visual FoxPro pe Web

Poate pentru mulți pare intrigant, dar Visual FoxPro 6.0 oferă o extensie ISAPI (*Internet Server Application Programing Interface*) suficient de puternică pentru dezvoltarea de aplicații pe Internet cu acces în timp real la baze de date. Vorbind despre acces real la date nu ne referim doar la tabele Visual FoxPro, ci și la alte tipuri de baze de date la care acesta se poate conecta (ORACLE, SQL-SERVER).

Temelia unei aplicații Internet – Visual FoxPro este constituită din utilitarul `FOXISAPI.dll` și `componentele COM` (*Common Object Model*). Acestea din urmă sunt biblioteci de clase publice, create în Visual FoxPro, ce pot fi utilizate de alte aplicații.

FoxISAPI este dedicat serverului de Internet MIIS (*Microsoft Internet Information Server*).

14.1. FoxISAPI și Servere COM – fundamentul aplicațiilor VFP pe Web

FoxISAPI reprezintă o bibliotecă de funcții (realizată de Microsoft) scrisă în C++, cu rol de aplicație **conector** între browserul-client care trimite o cerere și serverul COM căruia îi este adresată. Acesta din urmă va efectua diverse prelucrări (eventual cu implicații asupra unor colecții de date) și va trimite un răspuns sub formă de text `HTML`.

14.1.1. FoxISAPI. Noțiune și funcționalitate

Nucleul FoxISAPI este alcătuit din două fișiere, `foxisapi.dll` și `foxisapi.ini`, furnizate o dată cu pachetul Visual Studio, mai precis cu MSDN Library. Aceste fișiere pot fi regăsite în directorul de bază unde ați instalat MSDN-ul, pe calea: `...\MSDN98\98VSa\1033\SAMPLES\VFP98\Servers\Foxisapi` sau pe site-ul Microsoft.

Caracteristicile esențiale ce definesc FoxISAPI sunt[62]:

- **Extensie Web de tip Server Side**

`Foxisapi.dll` se execută în exclusivitate pe server, ca aplicație ISAPI în back-end-ul serverului de Internet MIIS (*Microsoft Internet Information Server*). Așadar, clienții nu vor avea nevoie de bibliotecile de bază ale Visual FoxPro pentru a rula aplicația. Avantajele majore al acestui tip de aplicație (Server Side) privesc, pe de o parte, disponibilitatea imediată a actualizărilor către toți clienții, iar pe de altă parte, securitatea sporită a datelor.

- **Interfață directă, de mare performanță, cu aplicațiile scrise în Visual FoxPro**

Eficiența sporită constă în faptul că FoxISAPI se conectează la serverul Web printr-un singur apel (pentru fiecare cerere a clientului) ce accesează în mod direct componenta COM și utilizează rezultatul acesteia prin înglobarea sa într-o pagină Web.

62. Rick Strahl, *Building applications with FoxISAPI*, http://www.west-wind.com/.

- **Utilizează Internet Server API (ISAPI)**

ISAPI este o extensie extrem de performantă, de tip multifir (*multi-threaded*), ce lucrează ca interfață la nivel de sistem de operare.

- **Utilizează COM pentru transferul mesajelor**

Pe lângă faptul că furnizează o interfață *low-level* către serverul IIS (*Internet Information Server*), ISAPI utilizează Common Object Model pentru a comunica cu aplicația Visual FoxPro. Pe de altă parte, aplicația VFP este compilată ca obiect COM și se comportă ca server pentru clientul FoxISAPI.

În figura 14.1 observăm arhitectura unei aplicații Web în Visual FoxPro.

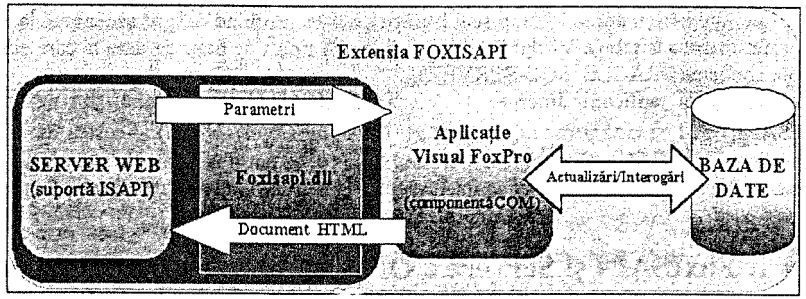


Figura 14.1. FoxISAPI – interfață între aplicația Visual FoxPro și serverul Web

Tabelul de mai jos prezintă comparativ cele două tipuri de biblioteci de funcții ce contribuie la crearea unei aplicații Visual FoxPro pe Web:

foxisapi.dll	Serverul COM Visual FoxPro
• Win32 DLL scris în C++ • Suportă execuții multifir (*multi-threaded*) • Aplicație de tip conector • Client COM	• COM EXE sau DLL • Monofir (*single-threaded* – un proces nu poate executa mai multe acțiuni simultan) • Aplicație ce accesează baze de date • Server COM

Sarcina de bază a extensiei FoxISAPI este aceea de a crea o instanță a componentei COM Visual FoxPro și de a invoca una dintre metodele acesteia. Indiferent dacă metoda respectivă execută sau nu anumite prelucrări, ea trebuie obligatoriu să returneze un text în format `HTML` care va fi transmis înapoi, de către FoxISAPI, serverului Web.

Astfel, în momentul în care clientul trimite o cerere de acces spre o componentă COM, serverul Web cere de fapt încărcarea mai întâi a bibliotecii `foxisapi.dll` și abia apoi, prin intermediul acesteia, are loc invocarea metodei solicitate. Mecanismul de care vorbim se declanșează printr-o legătură (*link*) URL de genul:

http://NumeServerWeb/scripts/foxisapi.dll/NumeServerVFP.Clasă.Metodă

Prin aceasta, serverul Web este determinat să încarce FoxISAPI în spațiul său de memorie. Pentru că FoxISAPI este Win32 DLL, se încarcă o singură dată și persistă pe toată durata execuției procesului apelant. Fiind de tip multifir, el poate prelua mai multe cereri simultan (dacă 10 clienți accesează același URL care se execută, să zicem, în 15 secunde, vom avea 10 procese care încearcă să acceseze același obiect COM). Din nefericire, o aplicație Visual FoxPro (deci și un server COM) este monofir (*single-threaded*) și nu poate rezolva simultan cereri multiple. S-ar părea, astfel, că `foxisapi.dll` ar trebui să țină evidența cererilor respective și să le trimită secvențial serverului COM. Totuși, nu este deloc așa: FoxISAPI poate crea instanțe multiple ale aceluiași server COM. Vom reveni asupra acestui aspect spre sfârșitul acestui capitol.

De asemenea, FoxISAPI este suficient de bine construit pentru a încărca (în urma unei cereri de acces) o singură dată componenta server în memoria internă, pentru ca apoi să rezolve următoarele cereri prin referință la acesta. Practic, prima cerere a unui client va fi rezolvată mai încet, datorită încărcării întregilor biblioteci de funcții VFP necesare execuției aplicației COM, în timp ce următoarele vor fi procesate extrem de rapid, pentru că vor accesa serverul gata încărcat în memorie.

Configurarea FoxISAPI constă, în primă fază, în copierea fișierelor `foxisapi.dll` și a fișierului de inițializare corespunzător `foxisapi.ini` într-un director accesibil serverului Web, de obicei: `C:\Inetpub\Scripts`.

14.1.2. Componente COM în Visual FoxPro

Componentele COM (*Common Object Model*) reprezintă de fapt biblioteci de clase ce încapsulează secvențe de cod care pot rezolva probleme comune mai multor tipuri de aplicații. Clasele respective sunt declarate publice și se înregistrează în regiștrii Windows, devenind accesibile oricărei aplicații care suportă COM. Accesul se realizează prin intermediul unor parametri pe baza cărora aplicația execută diverse prelucrări și returnează un rezultat intrepretat de procesul apelant.

Practic, în Visual FoxPro construim un server COM prin compilarea unui proiect ce cuprinde clase (în general de tip `CUSTOM`) declarate `OLE Public`. Doar acestor clase le va fi asociat un identificator unic global (`GUID`) în `Windows Registry` și vor fi accesibile altor aplicații (ca de exemplu FoxISAPI).

Pentru exemplificare, vom iniția proiectul `Test_COM` cu un singur program, prin care se definește o clasă cu numele `ComTest` ce conține metoda `PaginaStart` (listingul 14.1). Funcționalitatea acesteia constă în returnarea unui text `HTML` care să cuprindă data și ora la care a fost generat, precum și valorile curente ale parametrilor (despre aceștia vom discuta imediat).

Listing 14.1. Secvență de cod pentru crearea unei componente COM (`clasacom1`)

```
#DEFINE CR CHR(13)+CHR(10) && salt la rândul următor (pentru HTML)

DEFINE CLASS clasacom1 AS Custom OLEPUBLIC

FUNCTION PaginaStart
LPARAMETER pVariabile, pIniFile, pReleaseFlag
LOCAL text_html

*** HTTP header - necesar unor versiuni mai vechi de browsere
Text_html="HTTP/1.0 200 OK"+CR+;
     "Content-type: text/html"+CR+CR
*** construim textul in format HTML
text_html=text_html+;
  "<HTML><BODY>"+CR+;
  "<H1>Pagina TEST</H1><HR>"+CR+;
  "Acesta pagina a fost generata de  Visual FoxPro astazi: ";
  +DTOC(DATE())+" la ora :"+TIME()+ "<HR>"+CR+;
```

```
    +"Variabile : "+pVariabile+" /Fis. initializare:"+pIniFile+" /flag: "+str(pReleaseFlag)+CR+;
    "</HTML></BODY>"
  RETURN text_html
  ENDDEFINE
```

În cazul în care lucrăm în mod asistat, pentru a obține o clasă publică apelăm meniul `Class → Class Info` în `Class Designer` și bifăm opțiunea corespunzătoare (figura 14.2).

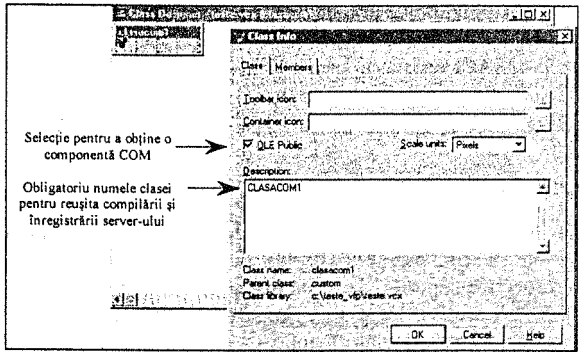


Figura 14.2. Construirea vizuală a unei clase `OLE Public`

După cum observăm și în figură, în cazul claselor create în mod asistat, este obligatoriu să mai introducem o dată numele clasei în zona de descriere: `Description`. Conform testelor efectuate, dacă nu procedăm astfel, obținem o componentă căreia, deși i se atribuie un identificator unic în `Windows Registry`, acesta nu se leagă de numele componentei, fiind în final inutilizabilă.

Atenție! Este extrem de important să înțelegem că o asemenea clasă (definită pentru a rula ca aplicație Web) conține în general două categorii de metode:

- **Metode ce vor fi accesate de serverul Internet** în urma unor cereri ale clienților. Aceste metode sunt apelate de fapt prin intermediul FoxISAPI, care le *trimite în mod implicit trei parametri*. Ca atare, orice metodă de acest tip va trebui să-i poată prelua, altminteri riscăm să obținem o eroare ce va duce la blocarea serverului.
- **Metode ce execută diverse prelucrări interne**, apelate exclusiv de alte metode ale clasei respective. Acestea vor avea un număr variabil de parametri (sau nu vor avea deloc), în funcție de necesități.

Semnificația parametrilor în cazul metodelor publice (conform numelor date în listingul 14.1) este următoarea:

- `pVariabile` – un șir de caractere ce cuprinde variabilele (parametrii actuali) furnizate de clientul Web pentru prelucrări specifice. O metodă poate fi apelată însă și fără parametri actuali, caz în care FoxISAPI transmite o valoare vidă. Este singurul parametru ce asigură interacțiunea directă cu utilizatorii aplicației;
- `pIniFile` – numele unui fișier `.ini` creat de `foxisapi.dll` la instanțierea unui server COM Visual FoxPro. Fiecare asemenea fișier de inițializare este creat în directorul `Scripts` (acolo unde se află `foxisapi.dll` și `foxisapi.ini`) și are un nume formalizat, de genul: `FOXnn.ini` (exemplu: `FOX55.ini`). Acesta va conține toate variabilele serverului, ale browserului și de sistem, cu posibilitatea de a extrage informații diverse (exemplu: adresa IP a browserului ce accesează metoda) prin intermediul funcției API `GetPrivateProfileString()`;
- `pReleaseFlag` – parametru de tip numeric *transmis prin referință*[63] de `foxisapi.dll`, prin care se specifică dacă instanța componentei COM persistă (rămâne rezidentă în memorie) după execuția metodei apelate. Instanța persistă după execuția metodei numai dacă `pReleaseFlag` are valoarea 0 (zero). În mod implicit, parametrul este transmis cu o valoare generată de FoxISAPI mai mare decât 0. Pentru performanțe optime, în multe cazuri acest parametru trebuie setat pe 0 în cadrul aplicației Web, altfel fiecare nouă cerere de acces va necesita reîncărcarea serverului în memorie și deci întârzierea răspunsului acestuia.

Un alt aspect de maximă importanță ce trebuie avut în vedere la crearea aplicației Web se referă la faptul că aceasta nu trebuie să genereze sub nici un motiv vreo formă de dialog cu utilizatorul de tip GUI (*Graphical User Interface*). Cu alte cuvinte, singura modalitate de răspuns a unei metode ce va fi apelată de serverul Web va fi întotdeauna un text în format `HTML`. Oricare din următoarele situații va duce la blocarea componentei COM:

- utilizarea funcției `MESSAGEBOX()` sau a comenzii `WAIT WINDOW`;
- încercarea de a deschide un formular (`DO FORM`);
- eroare netratată care va încerca să genereze un mesaj de tip GUI sau o fereastră de dialog de tip `File Open`.

O dată lămurite aceste aspecte, pasul următor constă în compilarea serverului. După cum observăm și în figura 14.3, în `Project Manager` vom utiliza butonul `Build` și opțiunea `Win32 executable`. Numele proiectului este `Test_com`, iar programul `Test` conține secvența de cod pentru definirea componentei `clasacom1` (listingul 14.1). Ca urmare, vom obține un fișier executabil, (eventual) cu același nume ca și al proiectului. Toate clasele publice definite în proiect vor fi înregistrate în `Windows Registry`, devenind astfel componente COM ce vor fi accesibile oricărei alte aplicații care suportă apeluri COM.

Atenție! Dacă nu ați instalat și Visual Studio 6.0 ServicePack 3, fereastra `Build Options` va arăta puțin diferit.

63. Mai multe despre modul de transmitere a parametrilor, în capitolul 4.

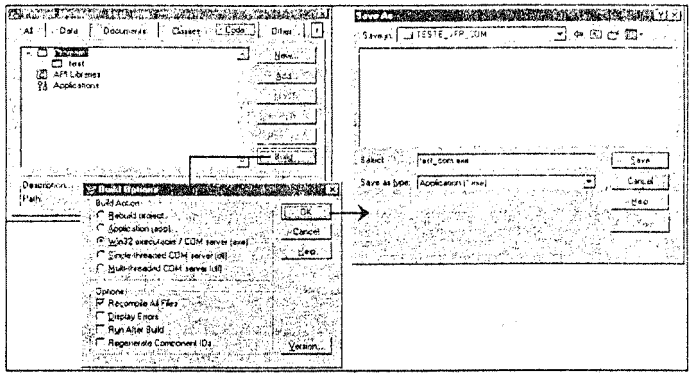


Figura 14.3. Compilarea serverului

În Visual FoxPro putem crea cele două mari tipuri de componente COM:

- ***proces extern (out-of-process)*** – obținem un fișier executabil (`.exe`) care se execută ca proces distinct față de aplicația ce-l va accesa. Comunicarea între cele două procese (clientul apelant și serverul apelat) este denumită comunicație *cross--process* (inter-procese);
- ***proces intern (in-process)*** – obținem o bibliotecă legată dinamic (DLL) care se execută în același spațiu de memorie ca și procesul-client (se comportă ca o procedură internă a procesului apelant).

Există avantaje și dezavantaje ale fiecăruia din perspectiva utilizării lor pe Web. Astfel, serverele de tip *in-process* pot fi caracterizate prin:

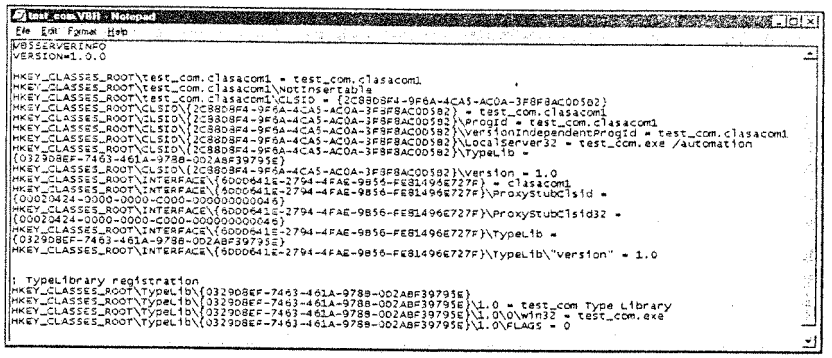
- interfața extrem de rapidă cu procesele-client apelante;
- blocarea lor duce în cele mai multe cazuri la blocarea serverului Web;
- nu pot fi descărcate complet o dată ce au fost încărcate de serverul MIIS. Aceasta poate deveni o problemă majoră dacă avem în vedere că orice aplicație trece prin mai multe versiuni intermediare, cu modificări dintre cele mai diverse, până la obținerea produsului final.
- sunt destul de greu de întreținut și depanat.

Componentele de tip *out-of-process* prezintă câteva avantaje demne de luat în considerare pentru dezvoltatorii de aplicații, fiind caracterizate prin:

- blocarea lor nu determină blocarea serverului de Web, MIIS, tocmai datorită faptului că se execută ca procese distincte;
- pot fi descărcate ușor folosind metoda `Reset` furnizată de FoxISAPI (ne vom referi la ea în finalul capitolului);
- oferă disponibilitate pentru instanțierea multiplă, caracteristică foarte importantă în rezolvarea simultană a mai multor cereri de acces la aceeași componentă;
- pot fi apelate via DCOM (*Distributed COM*). Doar componentele construite ca fișier executabil pot fi rezidente pe o stație oarecare în rețea și accesate de pe serverul de Internet;
- timp de acces mai mare datorită comunicării inter-procese (cu toate acestea, vorbim încă de milisecunde);

Este momentul să precizăm că, în testele noastre, am utilizat numai componente de tip *out-of-process*.

Compilarea unui proiect care conține clase definite `OLE Public` duce la crearea a trei fișiere cu același nume: unul de tip `.exe` sau `.dll`, unul de tip `.tlb` (*Type Library*) și altul de tip `.vbr` (fișier *Registry*). Acesta din urmă cuprinde identificatorii unici globali (GUID) pentru clasele publice precum și alte elemente specifice componentelor COM ce vor fi scrise în `Windows Registry` (figura 14.4).



```
VBSERVERINFO
VERSION=1.0.0

HKEY_CLASSES_ROOT\test_com.clasacom1 = test_com.clasacom1
HKEY_CLASSES_ROOT\test_com.clasacom1\NotInsertable
HKEY_CLASSES_ROOT\test_com.clasacom1\CLSID = {2C88D8F4-9F6A-4CA5-ACDA-3F8F8AC00582}
HKEY_CLASSES_ROOT\CLSID\{2C88D8F4-9F6A-4CA5-ACDA-3F8F8AC00582} = test_com.clasacom1
HKEY_CLASSES_ROOT\CLSID\{2C88D8F4-9F6A-4CA5-ACDA-3F8F8AC00582}\ProgId = test_com.clasacom1
HKEY_CLASSES_ROOT\CLSID\{2C88D8F4-9F6A-4CA5-ACDA-3F8F8AC00582}\VersionIndependentProgId = test_com.clasacom1
HKEY_CLASSES_ROOT\CLSID\{2C88D8F4-9F6A-4CA5-ACDA-3F8F8AC00582}\LocalServer32 = test_com.exe /automation
HKEY_CLASSES_ROOT\CLSID\{2C88D8F4-9F6A-4CA5-ACDA-3F8F8AC00582}\TypeLib =
{03290BEF-7463-461A-9788-002ABF39795E}
HKEY_CLASSES_ROOT\CLSID\{2C88D8F4-9F6A-4CA5-ACDA-3F8F8AC00582}\Version = 1.0
HKEY_CLASSES_ROOT\INTERFACE\{6DDD641E-2794-4FAE-9B56-FE81496E727F} = clasacom1
HKEY_CLASSES_ROOT\INTERFACE\{6DDD641E-2794-4FAE-9B56-FE81496E727F}\ProxyStubClsid =
{00020424-0000-0000-C000-000000000046}
HKEY_CLASSES_ROOT\INTERFACE\{6DDD641E-2794-4FAE-9B56-FE81496E727F}\ProxyStubClsid32 =
{00020424-0000-0000-C000-000000000046}
HKEY_CLASSES_ROOT\INTERFACE\{6DDD641E-2794-4FAE-9B56-FE81496E727F}\TypeLib =
{03290BEF-7463-461A-9788-002ABF39795E}
HKEY_CLASSES_ROOT\INTERFACE\{6DDD641E-2794-4FAE-9B56-FE81496E727F}\TypeLib\"Version" = 1.0

; TypeLibrary registration
HKEY_CLASSES_ROOT\TypeLib\{03290BEF-7463-461A-9788-002ABF39795E}
HKEY_CLASSES_ROOT\TypeLib\{03290BEF-7463-461A-9788-002ABF39795E}\1.0 = test_com Type Library
HKEY_CLASSES_ROOT\TypeLib\{03290BEF-7463-461A-9788-002ABF39795E}\1.0\0\win32 = test_com.exe
HKEY_CLASSES_ROOT\TypeLib\{03290BEF-7463-461A-9788-002ABF39795E}\1.0\FLAGS = 0
```

Figura 14.4. Fișier de tip `Registry` creat la compilarea serverului

După compilare, proiectul ne poate oferi câteva informații asupra serverelor COM pe care le înglobează. Astfel, dacă selectăm opțiunea `Project→Project Info` din meniul-sistem sau executăm un click-dreapta în `Project Manager`, vom obține o fereastră ca în figura 14.5.

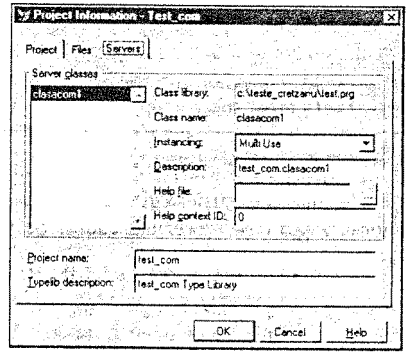


Figura 14.5. Informații privind clasele publice ale unui proiect compilat

Un aspect deosebit de important se referă la faptul că, în momentul compilării, Visual FoxPro realizează automat înregistrarea claselor publice. Avem însă și posibilitatea înregistrării/ștergerii componentelor COM în regiștrii Windows în mod independent.

Pentru înregistrarea unei componente `.exe` lansăm în execuție fișierul executabil cu parametrul `/REGSERVER`, iar pentru eliminarea ei utilizăm parametrul `/UNREGSERVER`.

Spre exemplu, `d:\Teste_Web\test_com.exe /unregserver` va determina eliminarea serverului `test_com` din `Windows Registry`.

În cazul componentelor `.dll` utilizăm utilitarul `REGSVR32` astfel:

```
REGSVR32 d:\Teste_Web\test_com.dll
```

va înregistra serverul `test_com` (dacă acesta ar fi fost `.dll`), pe când,

```
REGSVR32 /U d:\Teste_Web\test_com.dll
```

îl va elimina.

14.1.3. Testarea şi depanarea componentelor COM

Asemenea oricărei aplicaţii, înainte de a fi exploatată efectiv pe Web, o componentă COM trebuie mai întâi testată.

În acest scop, pentru exemplul nostru, vom introduce în fereastra de comandă a Visual FoxPro următoarele linii de cod:

```
X=CREATEOBJECT("test_com.clasacom1")
? x.paginastart("param1","param2",0)
RELEASE x
```

Prima linie va crea un obiect[64] cu numele „x" prin instanţierea clasei `clasacom1` ce se găseşte în biblioteca `test_com.exe`.

A doua linie va afişa direct pe ecranul VFP rezulatul metodei `paginastart`, care a fost apelată cu cei 3 parametri de test – bineînţeles, respectând tipul acestora.

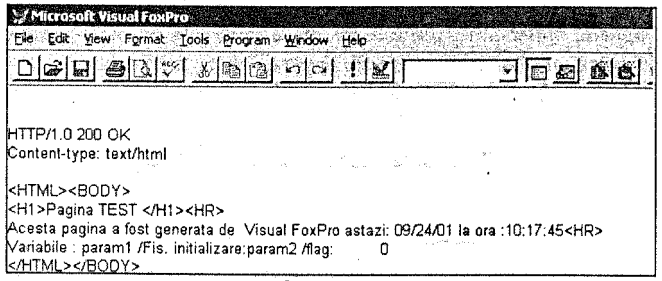


Figura 14.6. Rezultatul testării serverului COM `test_com.exe`

Crearea obiectului generează încărcarea fişierului executabil ce conţine clasa instanţiată, ca proces distinct în memorie. Figura 14.7 prezintă cadrul de pagină `Processes` din fereastra `Windows Task Manager`.

64. Vezi şi capitolul 11 – „Programare orientată-obiect în VFP".

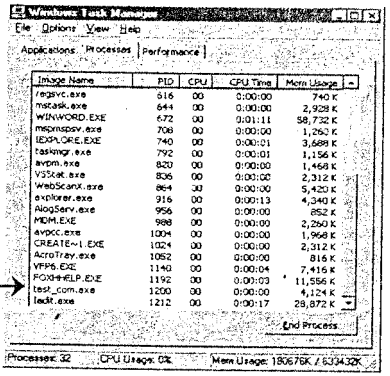


Figura 14.7. Server COM aflat în execuţie

Procesul va rămâne rezident în memorie până la execuţia unei instrucţiuni de descărcare (distrugerea obiectului „x" prin comanda `RELEASE x`) sau până la intervenţia directă cu butonul `End Process` în `Task Manager`. Această ultimă opţiune o vom folosi în cazul blocării aplicaţiei ca urmare a apariţiei unuia dintre evenimentele despre care am vorbit în subcapitolul anterior. Spre exemplu, apelul metodei `paginastart` fără parametri va duce la blocarea serverului:

```
x=CREATEOBJECT("test_com.clasacom1")
? x.paginastart
```

Ce se întâmplă de fapt? Ştim că orice parametru declarat, dacă nu are un corespondent actual, va avea implicit valoarea `.F.(False)`. Pentru că valorile parametrilor sunt utilizate, pe parcursul metodei, prin concatenarea lor cu un şir de caractere, neconcordanţa între tipul `caracter` şi tipul `logic` va genera o eroare. Erorile netratate generează un mesaj de avertizare de tip `GUI` care nu va ajunge niciodată la procesul apelant (mediul Visual FoxPro) şi va finaliza prin blocarea serverului în aşteptarea unui răspuns. În acest caz, singura modalitate de deblocare este butonul `End Process` din `Task Manager`.

14.1.4. Utilizarea serverelor COM pe Web. Configurarea drepturilor de acces

Pentru a putea utiliza o aplicaţie Visual FoxPro pe Web este necesară parcurgerea anumitor paşi:

- copierea fişierelor `foxisapi.dll` şi `foxisapi.ini` în directorul de scripturi al serverului Web (în general `c:\inetpub\scripts`);
- acordare drepturi de execuţie pe directorul respectiv;
- compilarea ca server *out-of-process* (fişier executabil) a aplicaţiei VFP ce conţine clase `OLE Public`;
- configurarea serverului COM;
- configurarea drepturilor de acces la date.

Aşadar, copierea celor două fişiere ce compun extensia FoxISAPI într-un director accesibil serverului de Internet nu este suficientă. Pe acest director trebuie să existe drepturi de execuţie care, pentru Microsoft Internet Information Server, se realizează astfel: în `Internet 'Services Manager`, expandăm nodurile până la directorul unde se află

`Foxisapi.dll` şi, în fereastra de setări `Properties`, selectăm pentru opţiunea `Execute Permissions` valoarea `Scripts and Executables` (figura 14.8).

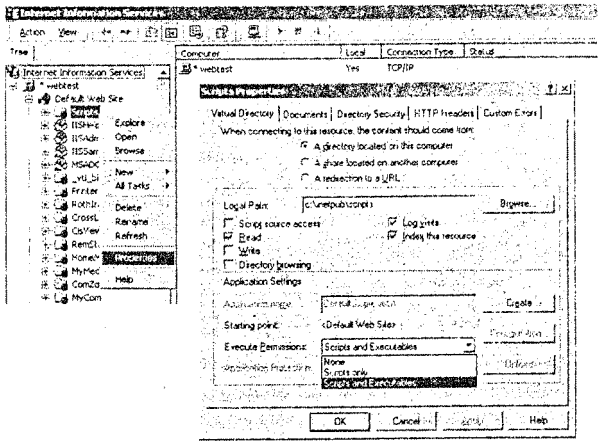


Figura 14.8. Acordarea dreptului de execuţie pentru directorul ce conţine fişierele `FoxISAPI`

În ceea ce priveşte configurarea componentelor COM, pentru a da drepturi de acces şi de execuţie conturilor utilizatorilor de Internet, vom folosi utilitarul Windows `DCOMCNFG` (lansarea în execuţie a acestuia o realizăm din meniul principal al Windows-ului, `Start→Run`). Acest utilitar listează toate componentele COM de tip `.exe` instalate pe computer (figura 14.9). Este imperios necesar să regăsim componenta noastră în lista respectivă; în caz contrar, serverul nu a fost compilat bine şi nu poate fi utilizat.

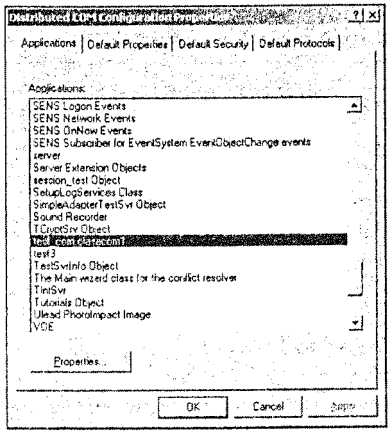


Figura 14.9. Utilitarul Windows `DCOMCNFG`

Configurarea poate fi realizată *la nivel global* (pentru toate componentele COM) şi la nivel individual (pentru fiecare în parte). În faza de testare este mai simplu să lucrăm la nivel global, astfel: în cadrul de pagină `Default Security` vom configura ***drepturile de acces*** (`Default Access Permissions`) şi ***drepturile de execuţie*** (`Default Launch Permissions`) pentru conturile-utilizator Windows NT/2000 specifice browserilor: `IUSR_NumeCalculator`; `IWAM_NumeCalculator` (figura 14.10):

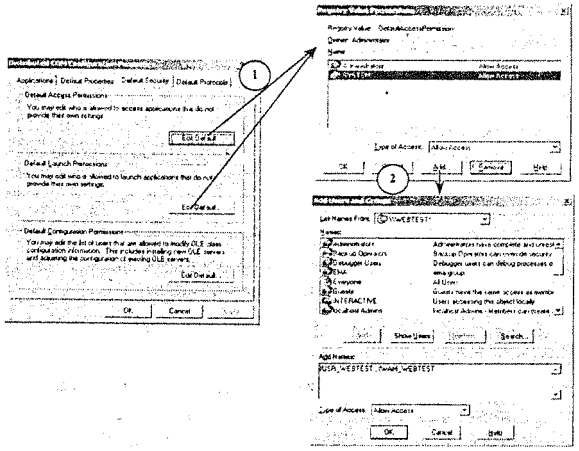


Figura 14.10. Configurarea globală a drepturilor de acces şi execuţie `COM`

Observăm, în figura de mai sus, că numele calculatorului (numele de reţea) în exemplul nostru este `webtest`. În final, fereastra de configurare va arăta ca în figura 14.11, cu precizarea implicită că numele serverului Web va fi, aproape sigur, altul.

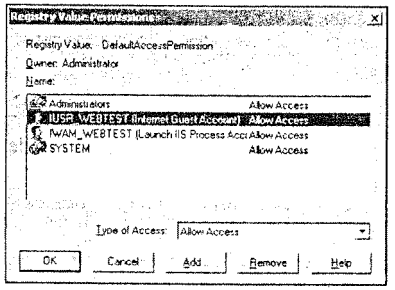


Figura 14.11. Drepturi de acces la componentele `COM` pentru conturile clienţilor Internet

În sfârşit, ultima configurare necesară pentru ca aplicaţia noastră să funcţioneze pe Web constă în ***acordarea drepturilor de acces şi modificare a datelor din baza de date***. Asta înseamnă că orice director care conţine fişiere ce vor fi accesate/modificate de componentele COM utilizate pe Web sau alte aplicaţii adiacente va trebui să permită acest lucru contului de utilizator NT/2000 specific browserilor: `IUSR_NumeCalculator` (figura 14.12).

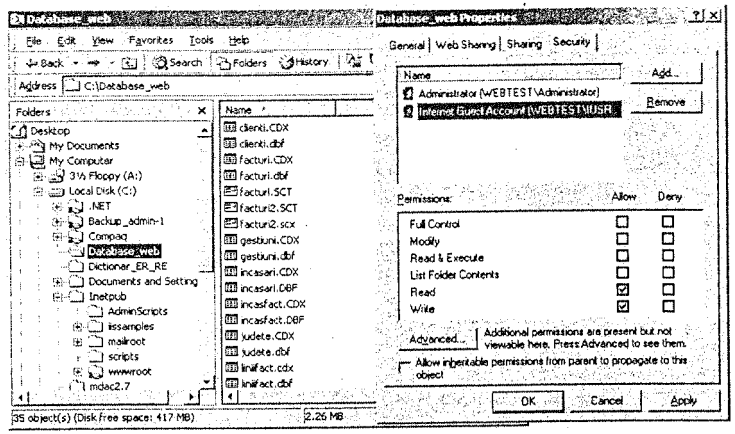


Figura 14.12. Configurarea drepturilor de acces/modificare pe directorul în care se află baza de date

Atenţie! Aceleaşi setări sunt obligatorii şi pentru directorul unde se află `foxisapi.dll` (`c:\Inetpub\Scripts`), datorită faptului că, la fiecare apel, acesta generează un fişier de iniţializare (`FOXnn.ini`) pe care apoi încearcă să-l şteargă. Ultima operaţiune nu va reuşi, bineînţeles, dacă nu acordăm drepturi de ştergere. În acest caz va trebui să efectuăm ştergerea periodică a fişierelor respective.

14.2. Prima aplicaţie Visual FoxPro pe Web

O dată ce am parcurs cu succes etapele premergătoare, putem trece la testarea efectivă a modului de funcţionare a unei componente COM pe Web. Ne vom opri în primul rând asupra serverului creat anterior: `test_com`. În acest sens, furnizăm browserului de Internet următoarea adresă:

http://NumeServerWeb/scripts/foxisapi.dll/test_com.clasacom1.paginastart

Un asemenea URL (*Universal Resource Locator*) va determina următoarele evenimente:

- serverul Web, identificat prin nume, încarcă în memorie biblioteca `foxisapi.dll`;
- `foxisapi.dll` lansează în execuţie `test_com.exe` şi invocă metoda `paginastart` prin instanţierea clasei publice `clasacom1`;
- metoda va returna un text în format `HTML`;
- `foxisapi.dll` preia răspunsul metodei, îl înglobează într-un document `HTML` şi îl trimite serverului Web;
- procesul `test_com` este oprit.

Rezultatul final, în fereastra browserului, ar trebui să fie cel din figura 14.13.

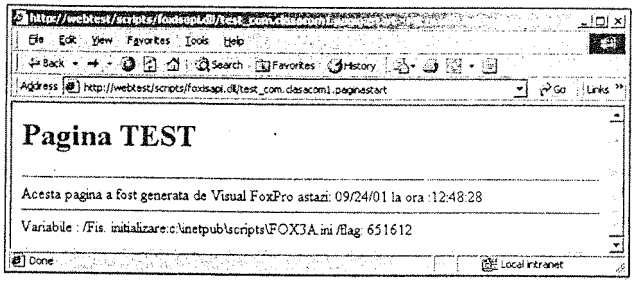


Figura 14.13. Testarea unei componente pe Web

După cum observăm, cei trei parametri de apel ai metodei sunt:

- primul parametru are o valoare vidă, pentru că nu am specificat nici o valoare în adresa furnizată browserului;
- al doilea parametru: `c:\inetpub\scripts\FOX39.ini`;
- al treilea parametru: `651612` (o valoare aleatoare generată de FoxISAPI, mai mare decât 0).

Pentru a furniza metodei anumite valori ce se doresc a fi prelucrate, completăm adresa de mai sus cu semnul „?" şi valorile dorite:

http://webtest/scripts/foxisapi.dll/test_com.clasacom1.paginastart?456%ETC%386

Rezultatul îl regăsim în figura 14.14.

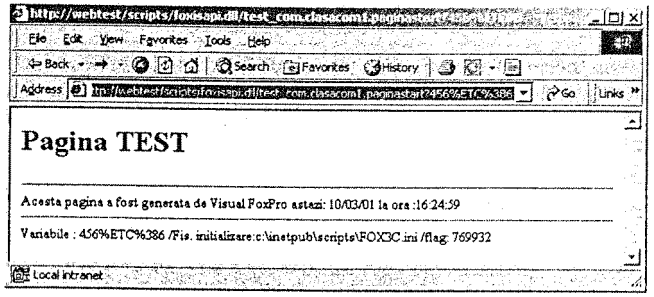


Figura 14.14. Utilizarea parametrilor în apelul metodelor pe Web

Exemplul, deşi luat la întâmplare, încearcă să evidenţieze folosirea unor caractere de delimitare în cazul în care dorim să transmitem mai multe valori ce vor fi prelucrate distinct.

14.3. Interogarea bazei de date. Interacțiunea cu utilizatorul

Interacțiunea cu utilizatorul constituie latura esențială a majorității aplicațiilor ce implică baze de date. Dacă în paragrafele anterioare am insistat pe tehnica de construire a unui modul Visual FoxPro utilizabil pe Web, în cele ce urmează vom evidenția modalități de conectare *on line* a utilizatorului (browserului) la date.

În acest context, pornind de la baza de date utilizată cu titlu de exemplu pe parcursul întregii cărți[65], presupunem că dorim să obținem un raport în format HTML care să prezinte situația vânzărilor pe clienți și produse (figura 14.15).

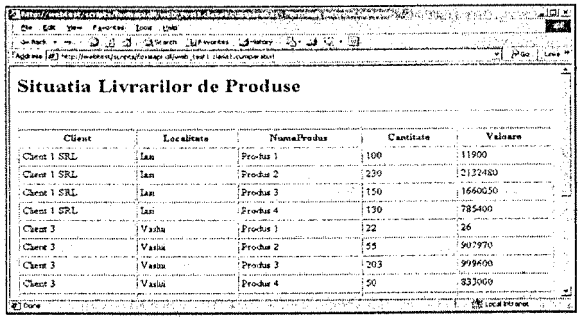


Figura 14.15. Raport HTML privind situația vânzărilor pe clienți și produse

În același timp, dorim să oferim utilizatorului posibilitatea de a filtra lista în funcție de produsul și/sau clientul dorit, astfel încât raportul să conțină fie toți clienții și produsele cumpărate de fiecare, fie un anumit produs și/sau un anumit client. În acest scop, vom construi un formular HTML ce va cuprinde un obiect de tip căsuță de text și altul de tip listă, în care să se poată introduce/selecteze un cod de client sau un produs anume (figura 14.16).

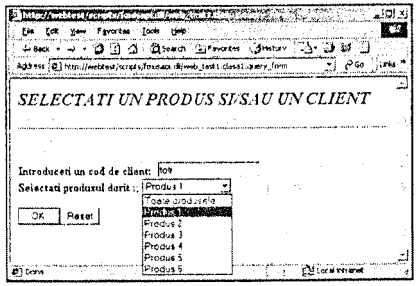


Figura 14.16. Formular HTML de preluare a opțiunilor utilizatorilor

65. Vezi capitolul 2.

Desigur, utilizarea unui obiect de tip căsuță de text pentru introducerea codului clientului este puțin cam forțată, scopul (pur didactic, de altfel) declarat fiind acela de a verifica, în aplicația VFP, dacă respectivul cod de client există în baza de date.

Dacă analizăm codul-sursă HTML[66] pentru formularul de mai sus (listingul 14.2) observăm că, pe de o parte, elementele listei (cu numele `Combo1`) trebuie generate dinamic în funcție de produsele existente la un moment dat în tabela `PRODUSE`, iar pe de altă parte, la apăsarea butonului `OK` va fi invocată metoda `Query_parametrizat` a componentei `clasa1`.

Listing 14.2. Sursa HTML a formularului din figura 14.16

```
<HTML>
<BODY>
<H2> <b><i> SELECTATI UN PRODUS SI/SAU UN CLIENT</i></b></H2><HR>
<FORM method="POST" ACTION="/scripts/foxisapi.dll/web_Test1.clasa1.query_parametrizat">
<p align="left"><BR>
<b>Introduceti un cod de client:  </b><input type="text" name="text1" size="20"
value="toti"><BR>
<b>Selectati produsul dorit:</b>; <SELECT NAME="COMBO1" size="1">
<OPTION SELECTED VALUE="toate"> Toate produsele
<OPTION VALUE=1> Produs 1
<OPTION VALUE=2> Produs 2
<OPTION VALUE=3> Produs 3
<OPTION VALUE=4> Produs 4
<OPTION VALUE=5> Produs 5
<OPTION VALUE=6> Produs 6
</SELECT> <BR><BR>
<input type="submit" name="OK" value="  OK  "><input type="reset">
</form>
</body>
```

Așadar, vom construi un server COM cu numele `web_test1` în care vom defini clasa publică denumită `clasa1` (listingul 14.3) ce va conține următoarele metode:

- `Init` – asigură deschiderea tabelelor necesare în momentul instanțierii clasei;
- `Query_form` – construiește un text HTML ce constituie codul-sursă al formularului din figura 14.16;
- `Query_parametrizat` – pe baza datelor de intrare (valorile obiectelor de tip `Input`) preluate din formular de primul parametru al metodei, realizează interogarea bazei de date și construiește codul-sursă pentru raportul din figura 14.15;
- `Destroy` – închide toate tabelele în momentul în care componenta este descărcată din memorie.

Atenție! Ca *regulă general valabilă pentru orice formular HTML*, atunci când clientul Web apasă un buton de tip `Submit` (exemplu: `OK`), `foxisapi.dll` transmite primului parametru al metodei invocate (în cazul nostru, parametrul `p_input_form`, metoda `Query_parametrizat`) un șir de caractere de genul: `text1=toti&COMBO1=1&OK=+++OK++++`.

66. Pentru simplificare, formularul poate fi creat cu ajutorul unui editor HTML, după care se va utiliza și modifica codul-sursă în cadrul aplicației VFP.

După cum se observă, şirul conţine numele tuturor obiectelor de tip `Input` ce se regăsesc pe formular, împreună cu valorile introduse de utilizator în ordinea în care sunt aşezate în pagina `HTML` şi delimitate prin semnul `&`. De asemenea, va conţine numele şi valoarea butonului de tip `Submit` ce a fost apăsat, iar spaţiile vor fi înlocuite cu semnul `+`.

Ca urmare, şirul de caractere va trebui să fie analizat pentru a extrage din el numai acele segmente ce contribuie la îndeplinirea obiectivului metodei.

Listing 14.3. Cod-sursă pentru definirea componentei `COM clasa1`

```
#DEFINE CR CHR(13)+CHR(10)

DEFINE CLASS clasa1 AS CUSTOM OLEPUBLIC

FUNCTION INIT
  SET DEFAULT TO  C:\database_web\
  USE produse IN 0 SHARED
  USE liniifact IN 0 SHARED
  USE facturi IN 0 SHARED
  USE localitati IN 0 SHARED
  USE clienti IN 0 SHARED
RETURN

FUNCTION QUERY_FORM
  LPARAMETER lcFormVars, lcIniFile, lnReleaseFlag
  LOCAL v_pagina_form1


v_pagina_form1= ;
      [<HTML>]+CR+;
      [<BODY>]+CR+;
      [<H2> <b><i> SELECTATI UN PRODUS SI/SAU UN CLIENT</i></b></H2><HR>]+CR+;
[<FORM                                                    method="POST"
ACTION="/scripts/foxisapi.dll/web_Test1.clasa1.query_parametrizat">]+CR+;
      [<p align="left"><BR>]+CR+;
[<b>Introduceti un cod de client:</b><input type="text" name="text1" size="20" value="toti"><BR>]+CR+;
      [<b>Selectati produsul dorit :</b>; <SELECT NAME="COMBO1" size="1">]+CR+;
      [<OPTION SELECTED VALUE="toate"> Toate produsele]+CR
** generam lista de elemente
  SELECT produse
  SCAN
      v_pagina_form1=v_pagina_form1+;
          [<OPTION VALUE=]+ALLTRIM(STR(produse.codpr))+[> ]+produse.denpr+CR
  ENDSCAN

  v_pagina_form1=v_pagina_form1+;
      [</SELECT> <BR><BR>]+CR+;
      [<input type="submit" name="OK" value="   OK    "><input type="reset">]+CR+;
      [</form>]+CR+;
      [</body>]+CR

RETURN v_pagina_form1

FUNCTION Query_parametrizat

LPARAMETER p_input_form,lcIniFile, lnReleaseFlag
LOCAL v_codcl,v_codpr,poz_delimitator,poz_egal
  ***       RETU p_input_form     && doar in faza de testare
  *** stim ordinea in care sunt asezate valorile introduse in sirul de caractere ce vine ca parametru
```



```
  poz_delimitator=AT('&',p_input_form)     && preluăm poziţia primului semn "&" în sirul de caractere
  v_codcl=LEFT(p_input_form,poz_delimitator-1)   && poz_delimitator-1 :eliminam '&'
  poz_egal=AT('=',v_codcl)
  v_codcl=RIGHT(v_codcl,LEN(v_codcl)-poz_egal)    && este de tip CHAR!!!!!

  p_input_form=RIGHT(p_input_form,LEN(p_input_form)-poz_delimitator)

  poz_delimitator=AT('&',p_input_form)
  v_codpr=LEFT(p_input_form,poz_delimitator-1)
  poz_egal=AT('=',v_codpr)
  v_codpr=RIGHT(v_codpr,LEN(v_codpr)-poz_egal) && este de tip CHAR!!!!


  v_text_html="HTTP/1.0 200 OK"+CR+;
      "Content-type: text/html"+CR+CR  && neaparat si o linie libera (al doilea CR)

  v_text_html=v_text_html+ ;
      [<HTML><BODY BGCOLOR="#FFFFFF">] + ;
      [<H1>Situatia Livrarilor de Produse</H1>] + [<p>]

  *** test exista cod client
  x=SEEK(v_codcl,'clienti','codcl')
  IF x=.F. AND v_codcl<>' toti '
      v_codcl='toti' && dam toti clientii daca a introdus un cod gresit
      v_text_html=v_text_html+;
          [<H3> Pentru toti clientii deoarece CODUL introdus NU este valid</H3><HR><p>]
  ENDIF

  ** construim cursorul in functie de valorile introduse
  DO CASE
  CASE v_codcl='toti' AND v_codpr='toate'
      SELECT denpr,codcl,SUM(cantitate) AS cant,SUM(1+proctva)*cantitate*pretunit AS valoare ;
          FROM facturi F,liniifact l,produse p ;
          WHERE l.nrfact=F.nrfact AND p.codpr=l.codpr;
          GROUP BY codcl,p.codpr;
          INTO CURSOR c1
  CASE v_codcl='toti'
      SELECT denpr,codcl,SUM(cantitate) AS cant,SUM(1+proctva)*cantitate*pretunit AS valoare ;
          FROM facturi F,liniifact l,produse p ;
          WHERE l.nrfact=F.nrfact AND p.codpr=l.codpr AND p.codpr=VAL(v_codpr);
          GROUP BY codcl,p.codpr;
          INTO CURSOR c1
  CASE v_codpr='toate'
      SELECT denpr,codcl,SUM(cantitate) AS cant,SUM(1+proctva)*cantitate*pretunit AS valoare ;
          FROM facturi F,liniifact l,produse p ;
          WHERE l.nrfact=F.nrfact AND p.codpr=l.codpr AND codcl=VAL(v_codcl);
          GROUP BY codcl,p.codpr;
          INTO CURSOR c1
  OTHERWISE && avem valori valide selectate pe ambele
      SELECT denpr,codcl,SUM(cantitate) AS cant,SUM(1+proctva)*cantitate*pretunit AS valoare ;
          FROM facturi F,liniifact l,produse p ;
          WHERE l.nrfact=F.nrfact AND p.codpr=l.codpr AND p.codpr=VAL(v_codpr);
                                  AND codcl=VAL(v_codcl);
          GROUP BY codcl,p.codpr;
          INTO CURSOR c1
  ENDCASE

  SELECT dencl,loc,denpr,cant,valoare;
      FROM clienti C,c1,localitati l;
```

```
      WHERE C.codcl=c1.codcl AND C.codpost=l.codpost;
      INTO CURSOR c2 ORDER BY dencl,denpr


v_text_html=v_text_html+ ;
      [<TABLE BGCOLOR=#EEEEEE CELLPADDING=4 BORDER=1 WIDTH=100%>]+CR+;
  [<TR BGCOLOR=#FFFFCC><TH>Client</TH><TH>Localitate</TH><TH>NumeProdus</TH>]+;
      [<TH>cantitate</TH><TH>valoare</TH></TR>]+CR

SCAN && cursorul C2 este in zona curenta de lucru
  *** Construim liniile tabelului ce va constitui raportul din figura 14.15
      v_text_html=v_text_html+ [<TR><TD>]+;
          ALLTRIM(IIF(EMPTY(c2.dencl),"<BR>",c2.dencl))+[</TD><TD>]+;
          ALLTRIM(IIF(EMPTY(c2.loc),"<BR>",c2.loc))+[</a></TD><TD>]+;
          ALLTRIM(IIF(EMPTY(c2.denpr),"<BR>",c2.denpr))+[</TD><TD>]+;
          ALLTRIM(STR(c2.cant))+[</TD><TD>]+;
          ALLTRIM(STR(c2.valoare))+[</TD></TR>]+CR

ENDSCAN

RETURN v_text_html

FUNCTION DESTROY
  CLOSE TABLES ALL
  CLOSE DATA ALL
RETURN

ENDDEFINE
```

Observăm în listingul de mai sus că pentru construirea textului HTML am utilizat paranteze pătrate în locul ghilimelelor. Acest lucru este necesar atât pentru sporirea lizibilității codului, cât și datorită faptului că textul generat conține el însuși ghilimele.

O dată construită și testată această componentă, o putem apela fie direct din browser (http://NumeServerWeb/scripts/foxisapi.dll/web_test1.clasa1.query_form), fie putem introduce un `link` în HomePage-ul serverului Web astfel:

```
<a HREF="/scripts/foxisapi.dll/web_test1.clasa1.query_form"> <i>INTEROGARI
PARAMETRIZATE</i></a> <p>
```

14.4. Formular de vizualizare/actualizare înregistrări

Orice aplicație Web care își propune să ofere acces în timp real la date conține măcar un formular de actualizare înregistrări în baza de date. Modalitatea de abordare a unei asemenea probleme constituie scopul exemplului ce-l vom prezenta în continuare.

Așadar, presupunem că se dorește realizarea unui formular de actualizare a facturilor. Având în vederea organizarea bazei de date VINZARI[67], știm că datele unei facturi sunt distribuite în două tabele, FACTURI și LINIIFACT, relaționate prin numărul de factură (câmpul Nrfact). Ca urmare, pe formular trebuie să regăsim la un moment dat o înregistrare din tabela FACTURI și toate înregistrările corespunzătoare (cu același număr de factură) din tabela LINIIFACT (figura 14.17).

67. Pentru structura tabelelor, vezi capitolul 2.

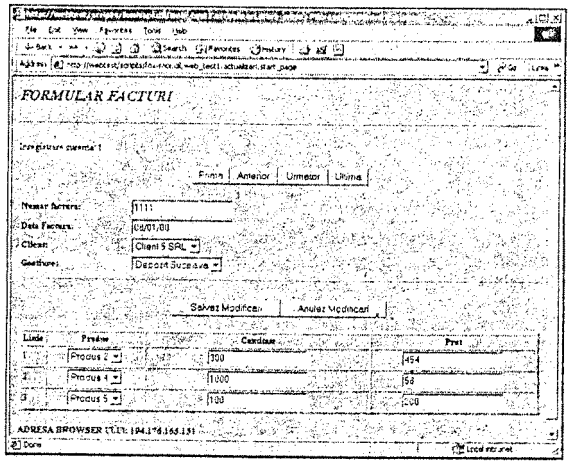


Figura 14.17. Formular pe Web pentru vizualizare/actualizare date în timp real

Codul-sursă HTML pentru formularul din figura de mai sus îl regăsim în listingul următor:

Listing 14.4. Codul-sursă HTML al formularului din figura 14.17

```
<HTML>
<BODY bgcolor="#C0C0C0">
<BODY>
<H2> <b><i>FORMULAR FACTURI</i></b></H2><HR><p>Inregistrare curenta:      1
<FORM method="POST" ACTION="/scripts/foxisapi.dll/web_Test1.actualizari.actiune">
<p align="center">
<input type="submit" name="primul" value="Prima"><input type="submit" name="anterior"
value="Anterior"><input type="submit" name="urmator" value="Urmator"><input type="submit"
name="ultimul" value="Ultima">
<input name="NR_INREGISTRARE" type="hidden" value = 1><p>
<table border="0" width="100%">
 <tr>
  <td width="21%"><b>Numar factura:</b></td>
  <td width="79%"><input type="text" name="TxtNrfact" size="20" value=    1111></td>
 </tr>
 <tr>
  <td width="21%"><b>Data Factura:</b></td>
  <td width="79%"><input type="text" name="TxtDatafact" size="20" value=08/01/00></td>
 </tr>
 <tr>
  <td width="21%"><b>Client:</b></td>
  <td width="79%"><SELECT NAME="CboClient" size="1">
<OPTION SELECTED VALUE=1005>Client 5 SRL
<OPTION VALUE=1001>Client 1 SRL
<OPTION VALUE=1002>Client 2 SA
<OPTION VALUE=1003>Client 3
<OPTION VALUE=1004>Client 4
<OPTION VALUE=1006>Client 6 SA
<OPTION VALUE=1007>Client 7 SRL
</SELECT></td>
 </tr>
 <tr>
```

```
    <td width="21%"><b>Gestiune:</b></td>
    <td width="79%"><SELECT NAME="CboGestiune" size="1">
  <OPTION SELECTED VALUE=002>Depozit Suceava
  <OPTION VALUE=001>Depozit Pacurari
  <OPTION VALUE=003>Sectie Pascani
  </SELECT></td>
   </tr>
  </table>
  <HR>
  <p align="center">
  <input type="submit" name="modifica" value="Salvez Modificari"><input type="reset"
value="Anulez Modificari"><p>
  <table border="3" width="100%">
   <tr>
    <td width="1%" align="center"><b>Linie</b></td>
    <td width="12%" align="center"><b>Produs</b></td>
    <td width="27%" align="center"><b>Cantitate</b></td>
    <td width="20%" align="center"><b>Pret</b></td>
   </tr>
  <tr>
  <td width="1%">
   <p align="left">1</td>
  <td width="12%">
   <p align="center"><SELECT NAME=CboProd1 size="1">
  <OPTION SELECTED VALUE=2>Produs 2
  <OPTION VALUE=1>Produs 1
  <OPTION VALUE=3>Produs 3
  <OPTION VALUE=4>Produs 4
  <OPTION VALUE=5>Produs 5
  <OPTION VALUE=6>Produs 6
  </SELECT></td>
    <td width="27%">
     <p align="center"><input type="text" name=TxtCant1 size="20" value=300></p>
    </td>
    <td width="20%">
     <p align="center"><input type="text" name=TxtPret1 size="20" value=454></p>
    </td>
   </tr>
  <tr>
  <td width="1%">
   <p align="left">2</td>
  <td width="12%">
   <p align="center"><SELECT NAME=CboProd2 size="1">
  <OPTION SELECTED VALUE=4>Produs 4
  <OPTION VALUE=1>Produs 1
  <OPTION VALUE=2>Produs 2
  <OPTION VALUE=3>Produs 3
  <OPTION VALUE=5>Produs 5
  <OPTION VALUE=6>Produs 6
  </SELECT></td>
    <td width="27%">
     <p align="center"><input type="text" name=TxtCant2 size="20" value=1000></p>
    </td>
    <td width="20%">
     <p align="center"><input type="text" name=TxtPret2 size="20" value=58></p>
    </td>
   </tr>
  <tr>
  <td width="1%">
   <p align="left">3</td>
```



```
  <td width="12%">
   <p align="center"><SELECT NAME=CboProd3 size="1">
  <OPTION SELECTED VALUE=5>Produs 5
  <OPTION VALUE=1>Produs 1
  <OPTION VALUE=2>Produs 2
  <OPTION VALUE=3>Produs 3
  <OPTION VALUE=4>Produs 4
  <OPTION VALUE=6>Produs 6
  </SELECT></td>
    <td width="27%">
     <p align="center"><input type="text" name=TxtCant3 size="20" value=100></p>
    </td>
    <td width="20%">
     <p align="center"><input type="text" name=TxtPret3 size="20" value=200></p>
    </td>
   </tr>
  </table>
  <p><H4> ADRESA BROWSER-ULUI: 194.176.165.151</h4>
  </form>
  </body>
```

La proiectarea unui formular ce va actualiza date în două tabele relaționate (părinte→copil) apar câteva probleme conceptuale suplimentare.

Astfel, pentru că mai mulți clienți pot accesa diverse înregistrări simultan, metoda ce asigură navigarea și/sau actualizarea înregistrărilor va trebui să dispună de un „punct de reper" care *să identifice în mod unic* înregistrarea pe care se află clientului x sau y la un moment dat (nu uităm că numărul de factură este modificabil). Un asemenea identificator unic poate fi considerat *numărul înregistrării*, care este alocat fizic la introducerea acesteia în tabelă și nu se modifică până la ștergerea definitivă a liniei respective. În exemplul de mai sus (listingul 14.4), obiectul de tip `Hidden` cu numele `Nr_inregistrare` introdus în formular are tocmai rolul de a stoca această valoare în vederea utilizării ei pentru navigare/actualizare.

Știm deja că la apăsarea unuia dintre butoanele de tip `Submit`, `foxisapi.dll` transmite ca parametru un șir de caractere ce cuprinde numele obiectelor de tip `Input` și valorile introduse sau selectate de utilizator. Ca urmare, în cazul modificării unei valori în tabela `Liniifact`, trebuie să existe un mecanism care să identifice înregistrarea pe care trebuie efectuată actualizarea.

O modalitate de rezolvare a acestei probleme ar fi utilizarea unei tabele derivate (*view*) cu numele `Vliniifact_1` (listingul 14.5) care să preia din tabela `Liniifact` doar liniile aferente facturii curente. Câmpurile modificabile din acest *view* vor fi apoi plasate pe formular prin intermediul unor obiecte a căror nume va urma o anumită codificare, astfel: `CboProdi`, `TxtCanti`, `TxtPreti`, unde `i` constituie numărul de linie (corespunzător câmpului `Liniifact.linie`) la care se referă obiectul. Pe baza acestui cod intern, metoda de actualizare va modifica valorile câmpurilor corespunzătoare (`codpr`, `cantitate`, `pretunit`) în tabela derivată, pe linia pe care s-a făcut actualizarea.

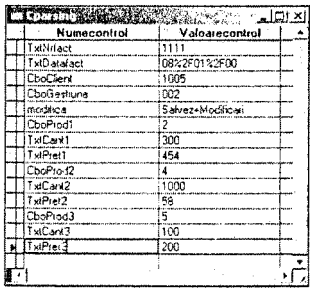
Un alt aspect foarte important se referă la faptul că, atunci când există mai multe butoane de tip `Submit` pe un formular, în șirul ce va fi transmis ca parametru se va regăsi *doar numele și valoarea butonului apăsat*. Astfel, în situația din figura 14.17, apăsarea butonului „Urmator" va trimite, ca prim parametru al metodei `Actiune`, un șir de caractere de genul:

urmator=Urmator&NR_INREGISTRARE=1&TxtNrfact=1111&TxtDatafact=08%2F01%2F00&CboClient=1005&CboGestiune=002&CboProd1=2&TxtCant1=300&TxtPret1=454&CboProd2=4&TxtCant2=1000&TxtPret2=58&CboProd3=5&TxtCant3=100&TxtPret3=200

Acest şir va trebui analizat în scopul extragerii valorilor ce corespund câmpurilor din cele două tabele. Este de la sine înţeles că şirul va avea o lungime variabilă, în funcţie de numărul liniilor unei facturi (numărul de linii din tabela derivată `vliniifact_1`).

Componenta COM, cu numele `Actualizari` (inclusă în acelaşi server `Web_test1`), prin care s-a realizat aplicaţia (listingul 14.6), cuprinde următoarele metode:

- `Init` – deschide tabelele necesare în mod partajat;
- `Start_Page` – este metoda apelată la prima cerere de acces a clientului Web la formular cu rol de poziţionare pe prima înregistrare în tabela `Facturi` şi de apel al metodei `Gen_Html`;
- `Gen_Html` – generează codul `HTML` al formularului (vezi şi listingul 14.4), în funcţie de poziţia curentă a clientului Web în tabela `Facturi`. Este apelată de metodele `Start_Page` şi `Acţiune` cu parametrul `p_adresaIP`, ce corespunde adresei browserului clientului (preluat din fişierul de iniţializare cu ajutorul funcţiei `GetPrivateProfileString()`);
- `Acţiune` – metodă apelată de formular ce analizează şirul primit ca parametru de la formular şi execută acţiunea solicitată de client: deplasare de la o înregistrare la alta sau actualizare date;
- `Comite_Modificari` – compară datele existente cu valorile primite ca parametru şi, dacă există diferenţe, „comite" modificările în baza de date;
- `Parsing_Input_Form` – analizează şirul de caractere primit ca parametru şi, în funcţie de poziţia delimitatorului `&`, extrage „blocurile" ce cuprind numele obiectului de tip `Input` şi valoarea sa (un exemplu de asemenea „bloc" extras din şirul de mai sus ar fi: `TxtCant3=100`). Fiecare segment de caractere este apoi transmis metodei `Repartizare`;
- `Repartizare` – populează un cursor cu două coloane, `numeobiect` şi `valoareobiect`, prin analiza unui segment furnizat de metoda `Parsing_Input_Form` şi extragerea a două subşiruri în funcţie de poziţia semnului „=" în şirul iniţial. În final vom obţine un cursor cu toate numele obiectelor de pe formular şi valorile lor (figura 14.18);
- `Destroy` – închide baza de date la descărcarea componentei din memorie.



	Numecontrol	Valoarecontrol
	TxtNrfact	1111
	TxtDatafact	08%2F01%2F00
	CboClient	1005
	CboGestiune	002
	modifica	Salvez+Modificari
	CboProd1	2
	TxtCant1	300
	TxtPret1	454
	CboProd2	4
	TxtCant2	1000
	TxtPret2	58
	CboProd3	5
	TxtCant3	100
▶	TxtPret3	200

Figura 14.18. Cursor obţinut în urma analizei şirului de caractere transmis ca parametru de formularul Web

Listing 14.5. Definiţia *view*-ului utilizat de componenta `Actualizari` pentru vizualizarea/actualizarea facturilor

```
nrfact_=1111 && variabila initializata cu o valoare aleatoare in scopul functionarii codului ce urmeaza
CREATE SQL VIEW vliniifact_1 AS ;
SELECT nrfact,linie,l.codpr,cantitate,pretunit,denpr ;
FROM liniifact l,produse p WHERE p.codpr=l.codpr AND nrfact=nrfact_ ORDER BY linie

DBSETPROP('vliniifact_1','view','tables','liniifact')

DBSETPROP('vliniifact_1.nrfact','field','keyfield',.T.)
DBSETPROP('vliniifact_1.linie','field','keyfield',.T.)

DBSETPROP('vliniifact_1.nrfact','field','updatename','liniifact.nrfact')
DBSETPROP('vliniifact_1.linie','field','updatename','liniifact.linie')
DBSETPROP('vliniifact_1.codpr','field','updatename','liniifact.codpr')
DBSETPROP('vliniifact_1.cantitate','field','updatename','liniifact.cantitate')
DBSETPROP('vliniifact_1.pretunit','field','updatename','liniifact.pretunit')

DBSETPROP('vliniifact_1.nrfact','field','updatable',.T.)
DBSETPROP('vliniifact_1.linie','field','updatable',.T.)
DBSETPROP('vliniifact_1.codpr','field','updatable',.T.)
DBSETPROP('vliniifact_1.cantitate','field','updatable',.T.)
DBSETPROP('vliniifact_1.pretunit','field','updatable',.T.)


DBSETPROP('vliniifact_1', 'view', 'SendUpdates', .T.)
```

BIBL. FACULTATEA UNIV. „AL. I. CUZA" IAŞI INFORMATICĂ

Listing 14.6. Program pentru definirea componentei `Actualizari` pentru obţinerea formularului Web din figura 14.17

```
#DEFINE CR CHR(13)+CHR(10)

DEFINE CLASS actualizari AS CUSTOM OLEPUBLIC

FUNCTION INIT
   **asiguram tratarea erorilor neprevazute
     PUBLIC a_fost_o_eroare
     a_fost_o_eroare='nu'
     ON ERROR a_fost_o_eroare=STR(ERROR())+"  " +MESSAGE()

     SET DEFAULT TO    c:\database_web\
     OPEN DATA vinzari SHARED
     USE produse IN 0 SHARED
     USE liniifact IN 0 SHARED
     USE facturi IN 0 SHARED
     USE localitati IN 0 SHARED
     USE clienti IN 0 SHARED
     USE gestiuni IN 0 SHARED

RETURN

FUNCTION START_PAGE
   LPARAMETER p_input_form,lcIniFile, lnReleaseFlag

   **Vom extrage (din fisierul .ini trimis ca parametru de foxisapi.dll) cu ajutorul functiei API
   **GetPrivateProfileString, adresa IP a  browser-ului client
```

```
    DECLARE INTEGER GetPrivateProfileString IN win32api STRING,STRING,STRING,;
    STRING @, INTEGER,STRING
    adresaIP = SPACE(500)
    nlen = GetPrivateProfileString("FOXISAPI","Remote Host","",@adresaIP,LEN(adresaIP),lcIniFile)
    adresaIP= LEFT(m.adresaIP,m.nlen)
    SELECT facturi
    GO TOP
RETURN this.gen_html(adresaIP)

FUNCTION ACTIUNE
    LPARAMETER p_input_form,lcIniFile, lnReleaseFlag
    *return p_input_form  && numai pentru teste

    LOCAL v_codcl,v_codpr,poz_delimitator,poz_egal
    lnReleaseFlag=0 && pastreaza serverul incarcat in memorie

    SET SAFETY OFF
    DECLARE INTEGER GetPrivateProfileString IN win32api STRING,STRING,STRING,;
       STRING @, INTEGER,STRING
    adresaIP = SPACE(500)
    nlen = GetPrivateProfileString("FOXISAPI","Remote Host","",@adresaIP,LEN(adresaIP),lcIniFile)
    adresaIP= LEFT(m.adresaIP,m.nlen)

    ** extragem numarul de inregistrare pe care se afla browser-ul
    x=AT('NR_INREGISTRARE=',p_input_form)
    trunc_sir=RIGHT(p_input_form,LEN(p_input_form)-(x+15))
    x=AT('&',trunc_sir)
    v_nr_inregistrare=VAL(LEFT(trunc_sir,x-1)) && v_nr_inregistrare trebuie transformat in tip numeric

    SELECT facturi
    GO v_nr_inregistrare

    a_fost_pozitionare=.F.
    * vom folosi operatorul $ pentru a verifica daca un sir de caractere se regaseste in altul.
    * Concret, vom căuta numele butonului ce a fost apasat pentru a sti ce trebuie sa facem
    DO CASE
      CASE "primul="$p_input_form
           a_fost_pozitionare=.T.
           GO TOP
      CASE "anterior="$p_input_form
           a_fost_pozitionare=.T.
           SKIP -1
           IF BOF()
                 SKIP 1
           ENDIF
      CASE "urmator="$p_input_form
           a_fost_pozitionare=.T.
           SKIP 1
           IF EOF()
                 SKIP -1
           ENDIF
      CASE "ultimul="$p_input_form
           a_fost_pozitionare=.T.
           GO BOTTOM
    ENDCASE
    IF a_fost_pozitionare=.T.
      RETURN THIS.gen_html(adresaIP)
    ENDIF

    DO CASE
```



```
        CASE 'modifica='$p_input_form  && utilizatorul a apasat butonul de comitere a modificarilor
             THIS.comite_modificari(p_input_form,v_nr_inregistrare)

             IF a_fost_o_eroare<>'nu'
                   RETURN a_fost_o_eroare
             ELSE
                   RETURN THIS.gen_html(adresaIP)
             ENDIF
    ENDCASE

RETURN

FUNCTION comite_modificari
    LPARAMETER p_sir_preluat,p_inreg_curenta
    ** ne vom pozitiona pe inregistrarea corespunzatoare in Facturi
    SELECT facturi
    GO p_inreg_curenta
    SET DATA TO vinzari
    nrfact_=facturi.nrfact

    IF !USED("vliniifact_1")
      USE vliniifact_1 IN 0
    ENDIF
    REQUERY("vliniifact_1")

    THIS.parsing_input_form(p_sir_preluat)

    ** cursorul Cparsing generat de metoda anterioara cuprinde numele obiectelor de pe formular si
    **valorile lor
    SELECT valoarecontrol FROM cparsing ;
      WHERE ALLTRIM(LOWER(numecontrol)) IN ("txtnrfact","txtdatafact","cboclient","cbogestiune");
      INTO ARRAY a_modif_facturi

    ** efectuam modificarile in Facturi daca e nevoie
    IF VAL(a_modif_facturi(1))<>facturi.nrfact
      REPLACE facturi.nrfact WITH VAL(a_modif_facturi(1))
    ENDIF
    IF VAL(a_modif_facturi(3))<>facturi.codcl
          REPLACE facturi.codcl WITH VAL(a_modif_facturi(3))
    ENDIF
    IF ALLTRIM(a_modif_facturi(4))<>ALLTRIM(facturi.gestiune)
          REPLACE facturi.gestiune WITH ALLTRIM(a_modif_facturi(4))
    ENDIF

    * efectuam modificarile in view, acolo unde este cazul, pe baza numarului de linie ce se regaseste
    * in  numele obiectelor de pe formular
    SELECT vliniifact_1
    a_fost_modificare=.F.
SCAN
  nr_rec=ALLTRIM(STR(RECNO()))
  **ordinea din cursor o stim ; si sigur avem tot atatea linii in pagina web cate in view
  SELECT valoarecontrol FROM cparsing ;
  WHERE ALLTRIM(LOWER(numecontrol)) IN ("cboprod"+nr_rec,"txtcant"+nr_rec,"txtpret"+nr_rec);
   INTO ARRAY a_modif_linie
  IF vliniifact_1.codpr<>VAL(a_modif_linie(1))
      REPLACE vliniifact_1.codpr WITH VAL(a_modif_linie(1))
      a_fost_modificare=.T.
  ENDIF
    IF vliniifact_1.cantitate<>VAL(a_modif_linie(2))
          REPLACE vliniifact_1.cantitate WITH VAL(a_modif_linie(2))
```

```
                    a_fost_modificare=.T.
    ENDIF
    IF vliniifact_1.pretunit<>VAL(a_modif_linie(3))
        REPLACE vliniifact_1.pretunit WITH VAL(a_modif_linie(3))
        a_fost_modificare=.T.
    ENDIF

    IF a_fost_modificare=.T.
      x=TABLEUPDATE()
      IF x=.F.
          TABLEREVERT()
      ENDIF
    a_fost_modificare=.F.
    ENDIF
ENDSCAN
RETURN

FUNCTION parsing_input_form
    LPARAMETER p_input

    poz_delimit_stg=0
    CREATE CURSOR cparsing(numecontrol c(50), valoarecontrol c(50))
    FOR i=1 TO LEN(p_input)
      IF SUBSTR(p_input,i,1)="&" && am ajuns la un delimitator
          ** vom extrage bucata de sir din stanga pana la delimitatorul anterior
          ** va iesi ceva de genul: numecontrol=valoare

      sir_extras=SUBSTR(p_input,poz_delimit_stg+1,i-1-poz_delimit_stg)

          * repartizam bucatile de sir din stg. si dr. egalului
      THIS.repartizare(sir_extras)
      poz_delimit_stg=i
      ENDIF
    ENDFOR
    ** metoda "repartizare" mai trebuie lansata o data pentru a extrage si ultimul Txtpret
    ** pentru ca acesta nu mai este urmat de un delimitator

    sir_extras=SUBSTR(p_input,poz_delimit_stg+1,LEN(p_input))
    THIS.repartizare(sir_extras)
RETURN

FUNCTION repartizare
    LPARAMETER p_sir
    ** separam sirul din stanga egalului de cel din dreapta
    poz_egal=0
    FOR m=1 TO LEN(p_sir)
      IF SUBSTR(p_sir,m,1)="="
          poz_egal=m
      ENDIF
    ENDFOR
    sir_dreapta=LEFT(p_sir,poz_egal-1)
    sir_stanga=RIGHT(p_sir,LEN(p_sir)-poz_egal)
    INSERT INTO cparsing VALUES (sir_dreapta,sir_stanga)
RETURN
```



```
FUNCTION gen_html
    LPARAMETERS p_adresaIP
    PUBLIC nrfact_
    nrfact_=facturi.nrfact
    SET DATA TO vinzari

    IF !USED("vliniifact_1")
      USE vliniifact_1 IN 0
    ENDIF
    REQUERY("vliniifact_1")

v_text_html=;
    [<HTML>]+CR+[<BODY bgcolor="#C0C0C0">]+CR+;
    [<BODY>]+CR+;
    [<H2> <b><i>FORMULAR FACTURI</i></b></H2><HR><p>Inregistrare curenta: ]+;
    STR(RECNO("facturi"))+CR+;
    [<FORM method="POST" ACTION="/scripts/foxisapi.dll/web_Test1.actualizari.actiune">]+CR+;
    [<p align="center">]+CR+;
    [<input type="submit" name="primul" value="Prima"><input type="submit" name="anterior" ;
    value="Anterior"><input type="submit"]+;
    [ name="urmator" value="Urmator"><input type="submit" name="ultimul" value="Ultima">]+CR+;
    [<input name="NR_INREGISTRARE" type="hidden" value = ]+ ;
     ALLTRIM(STR(RECNO("facturi")))+[><p>]+CR+;
    [<table border="0" width="100%">]+CR+;
    [  <tr>]+CR+;
    [    <td width="21%"><b>Numar factura:</b></td>]+CR+;
    [    <td width="79%"><input type="text" name="TxtNrfact" size="20"
value=]+STR(facturi.nrfact)+[></td>]+CR+;
    [  </tr>]+CR+;
    [  <tr>]+CR+;
    [    <td width="21%"><b>Data Factura:</b></td>]+CR+;
    [    <td width="79%"><input type="text" name="TxtDatafact" size="20" value=]+;
     DTOC(facturi.datafact)+[></td>]+CR+;
    [  </tr>]+CR+;
    [  <tr>]+CR+;
    [    <td width="21%"><b>Client:</b></td>]+CR+;
    [    <td width="79%"><SELECT NAME="CboClient" size="1"> ]+CR

    ** vom adauga obiectele de tip lista. Valoarea implicit selectata va fi valoarea curenta a clientului
    ** sau gestiunii din tabela Facturi

    v_codcl=facturi.codcl
    x=SEEK(v_codcl,"clienti","codcl") && cauta si pozitioneaza pe numele clientului din factura
curenta
    v_dencl=clienti.dencl
    v_gestiune=facturi.gestiune
    x=SEEK(v_gestiune,"gestiuni","gestiune")
    v_dengest=gestiuni.den_gest

    v_text_html=v_text_html+;
    [<OPTION SELECTED VALUE=]+ALLTRIM(STR(v_codcl))+[>]+ ALLTRIM(v_dencl)+CR

    SELECT clienti
    SCAN FOR codcl<>v_codcl
      v_text_html=v_text_html+;
      [<OPTION VALUE=]+ALLTRIM(STR(clienti.codcl))+[>]+ALLTRIM(clienti.dencl)+CR
    ENDSCAN
```

```
v_text_html=v_text_html+;
    [</SELECT></td>]+CR+;
    [  </tr>]+CR+;
    [  <tr>]+CR+;
    [    <td width="21%"><b>Gestiune:</b></td>]+CR+;
    [    <td width="79%"><SELECT NAME="CboGestiune" size="1"> ]+CR+;
    [<OPTION SELECTED VALUE=]+ALLTRIM(v_gestiune)+[>]+ ALLTRIM(v_dengest)+CR

    SELECT gestiuni
    SCAN FOR gestiune<>v_gestiune
      v_text_html=v_text_html+;
      [<OPTION VALUE=]+ALLTRIM(gestiuni.gestiune)+[>]+ALLTRIM(gestiuni.den_gest)+CR
    ENDSCAN

v_text_html=v_text_html+;
    [</SELECT></td>]+CR+;
    [  </tr>]+CR+;
    [</table>]+CR+;
    [<HR>]+CR+;
    [<p align="center">]+CR+;
    [<input type="submit" name="modifica" value="Salvez Modificari"><input type="reset" ;
     value="Anulez Modificari"><p>]+CR

    ** adaugam liniile facturii
    ** incepem cu generarea capului de tabel
    v_text_html=v_text_html+;
    [<table border="3" width="100%">]+CR+;
    [  <tr>]+CR+;
    [    <td width="1%" align="center"><b>Linie</b></td>]+CR+;
    [    <td width="12%" align="center"><b>Produs</b></td>]+CR+;
    [    <td width="27%" align="center"><b>Cantitate</b></td>]+CR+;
    [    <td width="20%" align="center"><b>Pret</b></td>]+CR+;
    [  </tr>]+CR
    SELECT vliniifact_1
    **continuam cu generarea liniilor efective preluate din view
    SCAN
      text_linie_de_adaugat=""
      SELECT codpr,denpr FROM produse WHERE produse.codpr<> vliniifact_1.codpr;
      INTO ARRAY elemente_combo
      nume_controlCombo="CboProd"+ALLTRIM(STR(RECNO()))
      nume_controlTxtcant="TxtCant"+ALLTRIM(STR(RECNO()))
      nume_controlTxtpret="TxtPret"+ALLTRIM(STR(RECNO()))
      text_elemente_combo="
      FOR I=1 TO ALEN(elemente_combo,1) && pentru toate liniile din matricea generata anterior
          text_elemente_combo=text_elemente_combo+;
[<OPTION
VALUE=]+ALLTRIM(STR(elemente_combo(I,1)))+[>]+ALLTRIM(elemente_combo(I,2))+CR
      ENDFOR
    text_linie_de_adaugat=;
    [<tr>]+CR+;
      [<td width="1%">]+CR+;
      [ <p align="left">]+ALLTRIM(STR(RECNO()))+[</td>]+CR+; && asta-i numarul liniei
      [<td width="12%">]+CR+;
      [ <p align="center"><SELECT NAME=]+nume_controlCombo+[ size="1"> ]+CR+;
    [<OPTION SELECTED VALUE=]+ALLTRIM(STR(vliniifact_1.codpr))+[>]+vliniifact_1.denpr+CR+;
    text_elemente_combo+; && are deja CR de la ultimul element generat din FOR
    [</SELECT></td>]+CR+;
    [    <td width="27%">]+CR+;
    [    <p align="center"><input type="text" name=]+nume_controlTxtcant+[ size="20" value=]+;
```



```
    ALLTRIM(STR(vliniifact_1.cantitate))+[></p>]+CR+;
    [  </td>]+CR+;
    [  <td width="20%">]+CR+;
    [ <p align="center"><input type="text" name=]+nume_controlTxtpret+[ size="20" value=]+;
    ALLTRIM(STR(vliniifact_1.pretunit))+[></p>]+CR+;
    [  </td>]+CR+;
    [ </tr>]+CR
    v_text_html=v_text_html+text_linie_de_adaugat
    ENDSCAN
    v_text_html=v_text_html+;
    [</table>]+CR+;
    [<p><H4> ADRESA BROWSER-ULUI: ]+p_adresaIP+[</h4>]+CR+;
    [</form>]+CR+;
    [</body>]
RETURN v_text_html
FUNCTION DESTROY
    CLOSE DATA ALL
RETURN
ENDDEFINE
```

Deşi, la prima vedere, metoda `GenHtml` pare complicată, ea se realizează foarte simplu dacă utilizăm un editor `HTML` pentru generarea formularului cu date de test pentru controale, după care intervenim în codul-sursă numai pentru a genera elementele listelor şi valoarea curentă pentru fiecare obiect în parte.

Observăm în listingul de mai sus şi utilizarea funcției API `GetPrivateProfileString()`, cu ajutorul căreia putem obține diverse informații din fişierul de inițializare creat de `foxisapi.dll` la fiecare execuție a unei componente COM. Conținutul unui asemenea fişier îl regăsim în listingul următor:

Listing 14.7. Fişier `.ini` creat de `foxisapi.dll` la instanțierea componentei `Actualizari`

```
[FOXISAPI]
Request Method=GET
Query String=
Logical Path=/web_test1.actualizari.start_page
Physical Path=C:\Inetpub\wwwroot\web_test1.actualizari.start_page
FoxISAPI Version=FoxISAPI v1.2
Request Protocol=HTTP/1.1
Referer=/scripts/foxisapi.dll
Server Software=Microsoft-IIS/5.0
Server Name=webtest
Server Port=80
Remote Host=194.176.165.152
Remote Address=194.176.165.152
[ALL_HTTP]
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
powerpoint, application/vnd.ms-excel, application/msword, */*
HTTP_ACCEPT_LANGUAGE=en-us
HTTP_CONNECTION=Keep-Alive
HTTP_HOST=webtest
HTTP_REFERER=http://webtest/
HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0; .NET CLR 1.0.2914)
HTTP_ACCEPT_ENCODING=gzip, deflate
[Accept]
image/gif=Yes
```

```
image/x-xbitmap=Yes
image/jpeg=Yes
image/pjpeg=Yes
application/vnd.ms-powerpoint=Yes
application/vnd.ms-excel=Yes
application/msword=Yes
*/*=Yes
[SYSTEM]
GMT Offset=7200
```

14.5. Managerul instanțelor multiple în FoxISAPI

După cum precizam și la începutul capitolului, orice aplicație Visual FoxPro este de tip monofir (*single-threaded*), pe când extensia ISAPI este multifir (*multi-threaded*). Ca urmare, singura modalitate de a beneficia de avantajul procesării simultane a mai multor cereri de acces la aceeași componentă COM constă în utilizarea unor instanțe multiple ale acesteia. Astfel, în timp ce o instanță a serverului preia o cerere ce determină prelucrări ceva mai substanțiale, o altă instanță poate prelua o cerere identică sau de un alt tip (care poate fi eventual rezolvată într-un timp mai scurt).

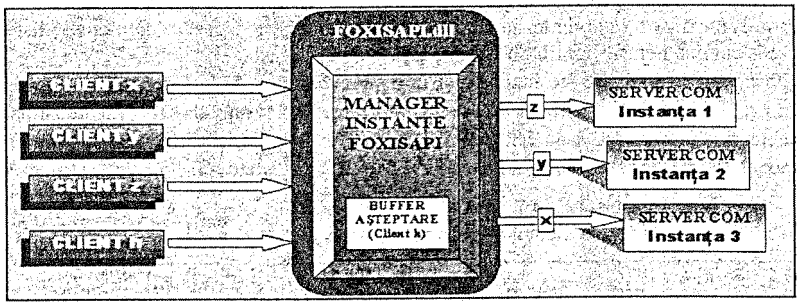


Figura 14.19. Rezolvarea simultană a mai multor cereri prin instanțierea multiplă a aceluiași server COM

Figura de mai sus prezintă o situație de management al mai multor cereri spre același server COM, pornind de la premisa că putem avea maxim trei instanțe ce pot exista simultan. Observăm că cererile ce aparțin primilor trei clienți (x, y și z) sunt trimise spre prelucrare în paralel tot atâtor instanțe ale serverului. În acest timp, o a patra cerere este menținută într-o zonă de memorie în așteptarea eliberării uneia dintre ele.

Pentru ca FoxISAPI să apeleze la managerul de instanțe, trebuie să adăugăm în fișierul de configurare `foxisapi.ini` două noi linii de cod: numele componentei COM între paranteze pătrate și numărul maxim de instanțe ce vor putea fi create. În urma acestei intervenții, fișierul de care vorbim va arăta ca în listingul 14.8:

Listing 14.8. Conținutul fișierului `Foxisapi.ini`

```
[foxisapi]
busytimeout=30
releasetimeout=40
statusurl=Status
reseturl=Reset
```



```
SingleModeUrl=SingleMode
MultiModeUrl=MultiMode
[web_test1.actualizari]
web_test1.actualizari=3
```

Extensia FoxISAPI dispune și de două metode proprii:

- `Status` – cu ajutorul ei putem afla informații despre instanțele serverelor COM aflate în execuție la un moment dat (figura 14.20);
- `Reset` – închide toate instanțele serverelor rezidente în memorie la un moment dat și resetează parametrii de stare.

Ele pot fi apelate astfel: http://webtest/scripts/foxisapi.dll/status.

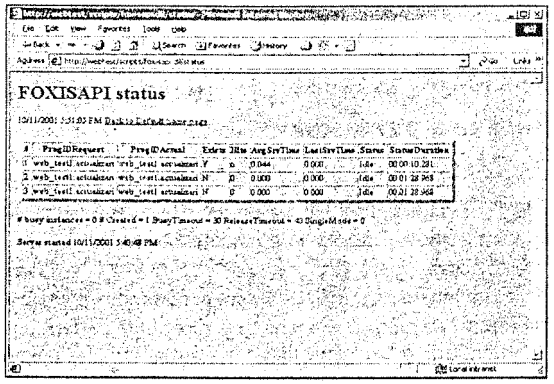


Figura 14.20. Metoda `Status` a extensiei `FoxISAPI`

Capitolul 15

Securitate în Visual FoxPro

Deşi este arhicunoscut faptul că Visual FoxPro nu oferă securitate de nici un fel asupra datelor, totuşi putem implementa „programatic" un sistem de protecţie la nivel de aplicaţie, folosind din plin securitatea sistemului de operare (Windows NT/2000).

15.1. Citire/scriere Windows Registry. O modalitate de stocare a parolelor

Visual FoxPro oferă un utilitar ce poate fi utilizat pentru creare/ştergere chei în `Windows Registry` şi citire/scriere de valori ale acestora. Utilitarul de care vorbim se prezintă sub forma unei biblioteci de clase cu numele `registry.vcx` şi poate fi găsit în directorul `...\Microsoft Visual Studio\Vfp98\FFC`. Aici vom regăsi definiţiile câtorva clase, cea care ne interesează în acest moment numindu-se chiar `Registry`.

Aşa cum probabil ştiţi, `Windows Registry` reprezintă de fapt un depozit de date în care sunt stocate informaţii vitale referitoare la configuraţia calculatorului şi a tuturor aplicaţiilor instalate. Regiştrii sunt organizaţi ierarhic, sub formă de arbore ce poate conţine chei, subchei şi opţiuni ale acestora. Fiecare cheie poate avea o valoare, iar fiecare opţiune este caracterizată prin nume, tipul datei stocate şi valoarea în sine.

Regiştrii pot fi editaţi cu utilitarul `Regedit` (în meniul principal al Windows-ului, `Start→Run→Regedit`).

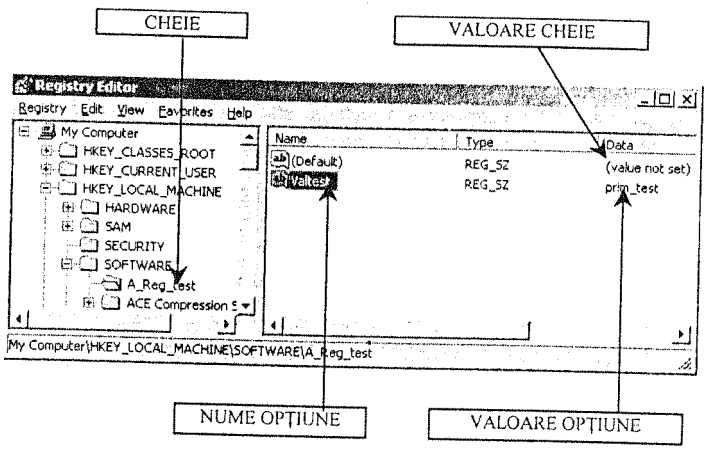


Figura 15.1. Utilitarul `Regedit`

Clasa `Registry` oferă posibilitatea de creare, citire şi editare chei şi opţiuni, în funcţie de necesităţi. Utilizarea sa presupune copierea fişierelor ce constituie biblioteca de clase `Registry` (`registry.vcx`, `registry.vct`, `registry.h`) într-un director al aplicaţiei şi adăugarea la proiect (butonul `ADD` în `ProjectManager→Classes`).

Tabelul 15.1 prezintă cele mai importante metode ale acesteia.

Tabelul 15.1. Metodele de bază ale clasei `Registry`

Metoda	Descriere
DeleteKey	Şterge o cheie din regiştri. Sintaxa: **DeleteKey(nUserKey, cKeyPath)** Argumente: *nUserKey* – valoare numerică *cKeyPath* – numele cheii (cuprinde şi calea pe arbore până la nivelul ei).
DeleteKeyValue	Şterge o opţiune din structura unei chei. Sintaxa: **DeleteKeyValue(cOptName, cKeyPath, nUserKey)** Argumente: *cOptName* – numele opţiunii *cKeyPath* – numele cheii (inclusiv calea pe arbore) *nUserKey* – valoare numerică.
GetRegKey	Returnează valoarea unei opţiuni din structura unei chei. Sintaxa: **GetRegKey(cOptName, cOptVal, cKeyPath, nUserKey)** Argumente: *cOptName* – numele opţiunii *cOptVal* – parametru, transmis prin referinţă[68] (@), ce va prelua valoarea asociată opţiunii respective *cKeyPath* – numele cheii (inclusiv calea pe arbore) *nUserKey* – valoare numerică.
SetRegKey	Modifică valoarea unei opţiuni din structura unei chei. Dacă opţiunea nu există, va fi creată şi i se va atribui valoarea specificată. Sintaxa: **SetRegKey(cOptName, cOptVal, cKeyPath, nUserKey, lCreate)** Argumente: *cOptName* – numele opţiunii *cOptVal* – valoarea ce va fi asociată opţiunii respective *cKeyPath* – numele cheii (inclusiv calea pe arbore) *nUserKey* – valoare numerică *LCreate* – `.T.` (`True`) – creează cheia (eventual şi cheile de nivel superior din cale) dacă nu există – `.F.` (`false`) sau lipsă – cheia trebuie să existe deja.

Notă. Toate aceste metode vor returna valoarea `0` dacă şi-au îndeplinit cu succes obiectivul şi `2` în caz contrar.

68. Vezi capitolul 4, pentru transmiterea parametrilor prin referinţă.

În Windows Registry, arborele de chei este organizat pornind de la cinci clase de bază (noduri rădăcină): HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_USERS, HKEY_LOCAL_MACHINE. Aceste noduri sunt furnizate metodelor clasei registry prin intermediul parametrului nUserKey, ce poate lua una din valorile prezentate în listingul 15.1. Din acest motiv, calea specificată prin CkeyPath pentru localizarea cheii nu va conține și nodul rădăcină.

Listing 15.1. Extras din fișierul-header: registry.h

```
* Registry roots
#DEFINE HKEY_CLASSES_ROOT       -2147483648 && BITSET(0,31)
#DEFINE HKEY_CURRENT_USER       -2147483647 && BITSET(0,31)+1
#DEFINE HKEY_LOCAL_MACHINE      -2147483646 && BITSET(0,31)+2
#DEFINE HKEY_USERS              -2147483645 && BITSET(0,31)+3
```

După cum se observă, pentru fiecare identificator de nod rădăcină este definită o constantă în fișierul-header al clasei registry.

Practic, pentru exemplul din figura 15.1, valorile parametrilor vor fi următoarele:

```
cKeyPath : "Software\A_Reg_test"
nUserKey : -2147483646  sau HKEY_LOCAL_MACHINE
```

Listingul 15.2 prezintă secvența de cod pentru crearea/citirea opțiunii cu numele *Valtest* și valoarea *„prim_test"* exemplificată în figura 15.1.

Listing 15.2. Procedura de scriere/citire Windows Registry

```
SET CLASSLIB TO registry
# INCLUDE registry.h
objtest=CREATEOBJECT("registry")
rez=objtest.SetRegkey("Valtest","prim_test","Software\A_Reg_test",HKEY_LOCAL_MACHINE,.T.)
IF rez<>0
    MESSAGEBOX("NU pot scrie in Registry",16,"Eroare")
ELSE
    Valreg=""
    objtest.GetRegkey("Valtest",@Valreg,"Software\A_Reg_test",HKEY_LOCAL_MACHINE)
    MESSAGEBOX(Valreg)
ENDIF
```

Metodele clasei Registry pot fi folosite în formulare în combinație cu proprietatea PasswordChar a unui obiect de tip Textbox, în scopul autentificării utilizatorilor unei aplicații sau pentru crearea/modificarea unei parole.

15.2. Obținerea numelui utilizatorului din rețea

Așa cum am văzut și în capitolul 14, în Visual FoxPro putem accesa funcții declarate în biblioteci externe (.dll) precum: Kernel32.dll, Gdi32.dll, User32.dll, Mpr.dll și Advapi32.dll. Astfel, într-o aplicație ce funcționează în rețea, putem obține numele stației de lucru și al utilizatorului curent cu ajutorul unor funcții API.

Înainte de a utiliza o funcție ce aparține unei biblioteci externe trebuie, în primul rând să-i cunoaștem numele și sintaxa (numărul și tipul parametrilor, care parametru este de

intrare și care de ieșire), iar în al doilea rând să o înregistrăm (instrucțiunea DECLARE-DLL).

În Windows există o multitudine de funcții API ce pot fi folosite pentru extinderea facilităților diverselor aplicații. O listă exhaustivă a acestora poate fi regăsită cu ajutorul utilitarului Microsoft Visual Studio 6.0 Tools→API Text Viewer (din meniul principal al Windows-ului), dacă acesta a fost instalat o dată cu pachetul Microsoft Visual Studio. Figura 15.2 prezintă fereastra principală a utilitarului de care vorbim după ce a fost încărcat fișierul win32api.txt folosind opțiunea File→Load Text File din meniul acestuia.

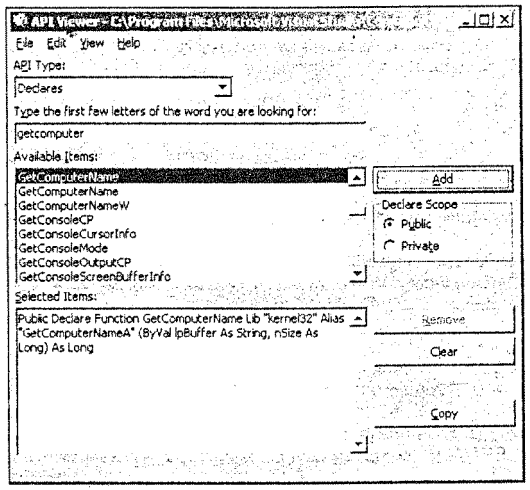


Figura 15.2. Vizualizarea listei de funcții API folosind utilitarul API Text Viewer

Observăm în figura 15.2 că, în zona Selected Items, regăsim sintaxa funcției selectate în format Visual Basic. Aceasta va trebui adaptată mediului Visual FoxPro.

Pentru mai multe informații privind o anumită funcție API, poate fi consultată documentația MSDN locală (dacă a fost instalată complet) sau de pe site-ul Microsoft (www.msdn.microsoft.com).

Așa cum precizam mai devreme, pentru a putea fi utilizată, funcția trebuie mai întâi declarată folosind sintaxa generală:

```
DECLARE [cFunctionType] NumeFunctie IN NumeDLL [AS AliasName]
  [cParamType1 [@] ParamName1,
     cParamType2 [@] ParamName2, ...]
```

Ca urmare, înregistrarea funcției selectate în figura 15.2 (GetComputerName) în Visual FoxPro se va realiza astfel:

```
  DECLARE GetComputerName IN   kernel32 AS GetComputerName
STRING @numecomp , LONG @complungime
```

Atenție! Numele funcțiilor trebuie scris identic cu cel specificat în API Viewer sau în documentație (litere mari și mici – *case-sensitive*), altfel funcția nu va fi regăsită în biblioteca respectivă și va genera o eroare. De asemenea, multe funcții necesită transmiterea parametrilor prin referință[69].

Listingul 15.3 prezintă secvența de cod pentru obținerea numelui calculatorului local, al utilizatorului curent și, eventual, numele de rețea al unui server ce conține un director partajat „*mapat*" local și în care se găsesc fișierele aplicației (`SET DEFAULT` stabilește calea spre discul virtual respectiv).

Listing 15.3. Funcția `AFLA_USER()`

```
** procedura pt. obtinere user, numecalculator, numeserver
LOCAL cCompBuf, cUsrBuf,nUsrLen, nCompLen

compbuf=SPACE(20)
usrbuf=SPACE(20)
usrlen=20
complen=20

DECLARE GetComputerName IN  kernel32 AS GetComputerName STRING @ compbuf , LONG
@complen
DECLARE GetUserName IN advapi32 AS GetUserName STRING @ usrbuf, LONG @ usrlen
DECLARE WNetGetConnection IN mpr.dll AS WNetGetConnection STRING  @ lpszLocalName,
STRING  @ lpszRemoteName, LONG  @ cbRemoteName

** invocam functiile
=GetUserName(@ usrbuf, @ usrlen)
username_=LEFT(ALLTRIM(usrbuf),LEN(ALLTRIM(usrbuf))-1)      && pentru că sirul returnat se
                                                            &&  termina cu nullstring
=GetComputerName(@ compbuf, @ complen)
computername_=LEFT(ALLT(compbuf),LEN(ALLT(compbuf))-1) && pentru că sirul returnat se
                                                            &&termina cu nullstring
**MESSAGEBOX(usrbuf,compbuf) &&numai pentru faza de testare

IF DRIVETYPE(SYS(5))=4 && suntem pe un disc virtual mapat din retea
                ** mai bine zis SET DEFAULT-UL ESTE PE O UNITATE MAPATA DIN RETEA!!!!

    lpszLocalName=SYS(5) &&directorul curent
    lpszRemoteName=SPACE(128)
    cbRemoteName=128
    =WNetGetConnection(@lpszLocalName,@lpszRemoteName,@cbRemoteName)

**  MESSAGEBOX(lpszRemoteName,STR(cbRemoteName)) && numai pt testare
    servername_=''                  && două apostroafe (valoare vida, tip caracter)
**  extragem numele server-ului dintr-un sir de genul " \\numeserver\sharedfolder "
    apare=0
    FOR i=1 TO LEN(lpszRemoteName)
        IF SUBSTR(lpszRemoteName,i,1)='\'
            apare=apare+1
            IF apare=3
                EXIT
            ENDIF
        ENDIF
        IF SUBSTR(lpszRemoteName,i,1)<>'\' AND i>=3
```

69. Vezi capitolul 4.


```
            servername =servername +SUBSTR(lpszRemoteName,i,1)
        ENDIF
    ENDFOR
ELSE && suntem pe harddisc sau alt disc local
    servername_=computername_
ENDIF

**MESSAGEBOX(servername_) && numai pt. testare

RETURN username_
```

15.3. Restricționarea editării tabelelor

Ideea mecanismului pe care vi-l propunem în continuare este relativ simplă. Se creează un director special denumit, spre exemplu ADMINISTRATOR, în care vom crea una sau mai multe tabele ce vor conține numele utilizatorilor preluate din Windows NT/2000 și drepturile de acces la fiecare tabelă. Prin configurări de securitate specifice sistemului de operare, toți utilizatorii, cu excepția administratorilor, vor avea doar drepturi de citire pentru aceste tabele (fișiere).

Prima tabelă cu această destinație, denumită chiar DREPTURI, este prezentată în listingul 15.4. Fiecare linie a acestei tabele specifică drepturile pe care le are un utilizator asupra unei tabele:

- dacă poate adăuga linii;
- dacă poate face modificări;
- dacă poate șterge înregistrări;
- intervalul calendaristic (datele și orele inițiale și finale) în care sunt valabile drepturile respective;
- dacă utilizatorul poate edita în weekend tabela.

Listing 15.4. Crearea și popularea unei prime tabele ce conține drepturi de acces

```
CREATE TABLE administrator\drepturi ( ;
  utilizator CHAR(25), ;
  tabela CHAR(20), ;
  inserare L, ;
  modificare L, ;
  stergere L, ;
  data_in DATETIME, ;
  data_fin DATETIME, ;
  weekend L ;
  )

INSERT INTO drepturi VALUES ('FotacheM', 'judete', .T., .T., .F.,
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('FotacheM', 'localitati', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('FotacheM', 'clienti', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('FotacheM', 'persoane', .F., .F., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('FotacheM', 'facturi', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('FotacheM', 'liniifact', .T., .T., .F., ;
```

```
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('FotacheM', 'produse', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('FotacheM', 'incasari', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('FotacheM', 'incasfact', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )

INSERT INTO drepturi VALUES ('CretuL', 'judete', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('CretuL', 'clienti', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('CretuL', 'persoane', .F., .F., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('CretuL', 'facturi', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('CretuL', 'liniifact', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('CretuL', 'produse', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('CretuL', 'incasari', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )
INSERT INTO drepturi VALUES ('CretuL', 'incasfact', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .F. )

INSERT INTO drepturi VALUES ('StrimbeiC', 'judete', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
INSERT INTO drepturi VALUES ('StrimbeiC', 'localitati', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
INSERT INTO drepturi VALUES ('StrimbeiC', 'persoane', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
INSERT INTO drepturi VALUES ('StrimbeiC', 'clienti', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
INSERT INTO drepturi VALUES ('StrimbeiC', 'persclienti', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
INSERT INTO drepturi VALUES ('StrimbeiC', 'facturi', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
INSERT INTO drepturi VALUES ('StrimbeiC', 'liniifact', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
INSERT INTO drepturi VALUES ('StrimbeiC', 'produse', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
INSERT INTO drepturi VALUES ('StrimbeiC', 'incasari', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
INSERT INTO drepturi VALUES ('StrimbeiC', 'incasfact', .T., .T., .F., ;
  {^2001/12/01 08:00:00}, {^2001/12/31 20:00:00}, .T. )
```

La deschiderea bazei de date (execuția procedurii stocate `SETUP`[70]), în afara operațiunilor descrise pe parcursul capitolelor precedente, se preia prin funcția `AFLA_USER()` (definită mai sus) numele utilizatorului Windows NT/2000 și, pentru fiecare tabelă, se inițializează o serie de variabile legate de autorizarea inserărilor, modificărilor, ștergerilor, zilelor/orelor (și weekend-ului) de lucru – vezi listingul 15.5.

70. Pentru conținutul procedurii, vezi capitolul 6 sau capitolul 13.

Listing 15.5. Inițializarea variabilelor ce conțin drepturi ale utilizatorului curent

```
*=============================================================================
  PROCEDURE SETUP     && varianta RETEA – si (mini) mecanism de securitate
*-----------------------------------------------------------------------------
SET DELETE ON
SET MULTILOCKS ON

* se verifica daca baze de date e deja deschisa SHARED
IF DBUSED('vinzari') AND !ISEXCLUSIVE('vinzari',2)
  * e-n regula
ELSE
      MESSAGEBOX('Baza de date VINZARI nu este deschisa partajabil ')
ENDIF

PUBLIC utilizator_
utilizator_ = AFLA_USER()

* se incearca sa se deschida SHARED toate tabelele BD
PUBLIC vTabele
DIME vTabele(5,1)
ADBOBJECTS(vTabele, "TABLE")

tab_ = 'drepturi'
IF !deschidere_tabele (UPPER(ALLT(tab_)))
  MESSAGEBOX('Cu durere in glas, aplicatia nu poate fi lansata !')
  QUIT
ENDIF

LOCAL i
FOR i = 1 TO ALEN(vTabele)
  tab_ = vTabele(i)
  * apelul functiei care incearca deschiderea SHARED a tabelei

  IF !deschidere_tabele (UPPER(ALLT(tab_)))
      MESSAGEBOX('Cu durere in glas, aplicatia nu poate fi lansata !')
      QUIT
  ENDIF

  * daca exista macar un index, primul (dintre indecsi) devine principal
  SELECT (tab_)
  IF TAGCOUNT() > 0
      SET ORDER TO TAG 1
  ENDIF
  PUBLIC trg_ins_&tab_, trg_upd_&tab_, trg_del_&tab_
  PUBLIC ins_&tab_, upd_&tab_, del_&tab_, data_in_&tab_, data_fin_&tab_, weekend_&tab_

  * se incarca drepturile utilizatorului pentru tabela curenta
  SELECT drepturi
  LOCATE FOR utilizator = utilizator_ AND UPPER(tabela) = UPPER(tab_)
  IF EOF()
      ins_&tab_ = .F.
      upd_&tab_ = .F.
      del_&tab_ = .F.
      data_in_&tab_ = .F.
      data_fin_&tab_ = .F.
      weekend_&tab_ =      .F.
  ELSE
      ins_&tab_ = drepturi.inserare
      upd_&tab_ = drepturi.modificare
```

```
      del &tab  = drepturi.stergere
      data_in_&tab_ = drepturi.data_in
      data_fin_&tab_ = drepturi.data_fin
      weekend_&tab_ =      drepturi.weekend
  ENDIF
ENDFOR

PUBLIC vRelatii, vCopii, vParinti, vTaguriCopil, vTaguriParinte
DIME vRelatii(4,4), vCopii(1), vParinti(1), vTaguriCopil(1), vTaguriParinte(1)
ADBOBJECTS(vRelatii, 'Relation')
* prima coloana din vRelatii - tabela COPIL
* a doua coloana din vRelatii - tabela PARINTE
* a treia coloana din vRelatii - index (tag) COPIL
* a patra coloana din vRelatii - index(tag) PARINTE
DIME vCopii(ALEN(vRelatii,1)), vParinti(ALEN(vRelatii,1)), ;
  vTaguriCopil(ALEN(vRelatii,1)), vTaguriParinte(ALEN(vRelatii,1))
FOR i = 1 TO ALEN(vRelatii,1)
  vCopii(i) = vRelatii(i,1)
  vParinti(i) = vRelatii(i,2)
  vTaguriCopil(i) = vRelatii(i,3)
  vTaguriParinte(i) = vRelatii(i,4)
ENDFOR

** la fiecare iesire din VFP se salveaza o copie a procedurilor stocate
ON SHUTDOWN DO salvare_proceduri_stocate

ENDPROC
*=============================================================================
```

După cum spuneam la începutul paragrafului, mino-mecanismul de securitate prezentat are la bază declanşatoarele. Astfel, cele trei declanşatoare apelează funcţia de autorizare a editării. Pentru exemplificare, în listingul 15.6 este prezentată prima parte a declanşatorului general de inserare în care se apelează funcţia stocată `Drepturi()`.

Listing 15.6. Inserarea secvenţei de autorizare în declanşatorul de inserare

```
*-------------------------------------------------------------
* declansator general pentru inserare
*-------------------------------------------------------------
PROCEDURE trg_ins
parameter tabela__
* MESSAGEBOX('trg Inserare '+tabela__)
LOCAL i, copil_, tagcopil_, nrindexcopil_, cheiecopil_, valoare_cheiecopil_,;
  tabela_, curent_, parinte_, tagparinte_, nrindexparinte_, ;
  cheieparinte_, valoare_cheieparinte_, sir_valoare_cheiecopil_

tabela_ = allt(upper(tabela__))
IF TYPE('vTabele') = "U" OR TYPE('vRelatii') = "U"
  do setup
ENDIF
trg_ins_&tabela_ = .t.

IF !drepturi(tabela_, 'INS')    ( ! – operator logic echivalent cu NOT)
  RETURN .F.
ENDIF

*
* restul declansatorului rămâne identic
*
```


Analog se inserează secvenţa `IF !drepturi(tabela_, 'UPD')` şi `IF !drepturi (tabela_, 'DEL')` în declanşatoarele generale de modificare, respectiv ştergere. Procedura stocată `Drepturi()` (de fapt, funcţie) este cea din listingul 15.7.

Listing 15.7. Funcţia stocată `drepturi`

```
*****************************************
PROCEDURE drepturi
*****************************************
PARAMETER tabela_, operatiune_
LOCAL var_
var_ = operatiune_+'_'+tabela_

* se testeaza daca utilizatorul este autorizat sa faca acest gen de operatiune
IF ! &var_
  MESSAGEBOX('Regret, dar in aceasta tabela nu puteti sa faceti '+;
      IIF(operatiune_='INS', 'inserari !', IIF(operatiune_='UPD', 'modificari !', 'stergeri !')))
  RETURN .F.
ENDIF

* sunt permise modificari in aceasta zi/ora ?
IF !BETWEEN(DATETIME(), data_in_&tabela_, data_fin_&tabela_)
  MESSAGEBOX('Din pacate, nu aveti autorizatia sa lucrati cu aceasta tabela la aceasta data si ora!')
  RETURN .F.
ENDIF

* daca e o zi din weekend, se face verificarea de rigoare ?
IF INLIST(CDOW(DATE()), 'Saturday', 'Sunday')
  IF !weekend_&tabela_
      MESSAGEBOX('Din pacate, nu aveti autorizatia sa lucrati cu aceasta tabela in weekend !')
      RETURN .F.
  ENDIF
ENDIF

ENDPROC
*****************************************
```

Sunt verificate, pe rând, următoarele drepturi:

- operaţiunea (inserarea, modificarea sau ştergerea) asupra tabelei respective de către utilizatorul curent;
- încadrarea datei şi orei modificării (`DATETIME()`) în intervalul alocat utilizatorului şi tabelei;
- dacă ziua curentă este o sâmbătă sau duminică, se verifică dacă sunt permise editări în weekend.

15.4. Autorizarea editărilor anumitor linii

Paragraful anterior a fost unul dedicat autorizărilor „pe verticală", în ideea de a încropi un „administrator" primitiv al bazelor de date VFP. În continuare ne vom avânta la un exemplu de restricţionare a accesului numai la anumite linii ale unei tabele. Concret, să presupunem că firma realizează facturarea on-line şi că s-a convenit ca fiecare client să fie încredinţat unui utilizator anume. Altfel spus, datele generale şi facturile legate de un client pot fi editate numai de unul sau mai mulţi utilizatori specificaţi.

Mai întâi creăm o a doua tabelă de drepturi, AUTORIZ_CL – vezi listingul 15.8.

Listing 15.8. Tabela alocărilor clienți-utilizatori

```
CREATE TABLE administrator\autoriz_cl ( :
  utilizator CHAR(25), ;
  codcl NUMERIC(6) ;
  )

INSERT INTO autoriz_cl VALUES ('CretuL', 1001)
INSERT INTO autoriz_cl VALUES ('CretuL', 1002)
INSERT INTO autoriz_cl VALUES ('CretuL', 1003)
INSERT INTO autoriz_cl VALUES ('CretuL', 1004)
INSERT INTO autoriz_cl VALUES ('CretuL', 1005)
INSERT INTO autoriz_cl VALUES ('CretuL', 1006)
INSERT INTO autoriz_cl VALUES ('CretuL', 1007)
INSERT INTO autoriz_cl VALUES ('CretuL', 1008)

INSERT INTO autoriz_cl VALUES ('FotacheM', 1001)
INSERT INTO autoriz_cl VALUES ('FotacheM', 1002)
INSERT INTO autoriz_cl VALUES ('FotacheM', 1003)
INSERT INTO autoriz_cl VALUES ('FotacheM', 1004)

INSERT INTO autoriz_cl VALUES ('StrimbeiC', 1005)
INSERT INTO autoriz_cl VALUES ('StrimbeiC', 1006)
INSERT INTO autoriz_cl VALUES ('StrimbeiC', 1007)
INSERT INTO autoriz_cl VALUES ('FotacheM', 1008)
```

Firește, se putea avansa pe ideea unei granularități mai fine, în sensul că fiecărei combinații utilizator-client i-ar putea fi specificate drepturi de inserare, modificare, ștergere sau intervale de editare diferite.

Pentru atingerea scopului propus, adăugăm în procedura stocată Setup comenzile din listingul 15.9. Se declară public vectorul vClienti, în care se încarcă numai codurile clienților la care utilizatorul curent este autorizat să facă modificări (datele din tabelele CLIENTI și FACTURI). A fost preferat vectorul pentru a se asigura o viteză rezonabilă de lucru.

Listing 15.9. Secvența adăugată la finalul procedurii stocate Setup

```
** codurile clientilor pentru care poate opera modificari utilizatorul curent
PUBLIC vClienti(1)
SELECT codcl FROM autoriz_cl INTO ARRAY vClienti WHERE utilizator = utilizator_

ENDPROC
```

Ultima modificare este cea a funcției stocate Drepturi(), în care se adaugă liniile din listingul 15.10. Verificarea se produce la modificarea și ștergerea unei linii din CLIENTI și la orice operațiune derulată asupra tabelei FACTURI.

Listing 15.10. Liniile de cod ce se adaugă funcției stocate Drepturi()

```
*
* ...
*
* daca este vorba de un declansator al tabelei FACTURI sau CLIENTI ,
* se verifica daca utilizator curentul are dreptul sa modifice date legate de acest clienti
```



```
IF tabela  = 'CLIENTI' OR tabela  = 'FACTURI'
  LOCAL codcl_
  DO CASE
     CASE  operatiune_ = 'INS' OR tabela_ = 'FACTURI'
        codcl_ = facturi.codcl
     CASE  operatiune_ = 'UPD' OR tabela_ = 'FACTURI'
        codcl_ = OLDVAL('codcl', 'facturi')
     OTHERWISE
        codcl_ = codcl
  ENDCASE
  IF ASCAN(vClienti, codcl_) = 0
     MESSAGEBOX('Nu puteti edita date (inclusiv facturi) pentru acest client !')
     RETURN .F.
  ENDIF
ENDIF

ENDPROC
**********************************************
```

Capitolul 16

Aplicații VFP cu servere de baze de date

16.1. Specificul aplicațiilor cu servere de baze de date

Sistemele informaționale economice de anvergură sunt clădite de regulă în jurul unui server de baze de date puternic, indiferent dacă datele gestionate în acest mod sunt disponibile în medii client/server sau pe Web.

Bazele de date implementate în astfel de medii sunt proiectate respectând cu strictețe anumite „canoane" privind modurile de organizare și sunt accesibile prin limbaje și structuri conforme cât mai mult posibil standardelor consacrate în domeniu. Caracteristicile generale ale unor astfel de sisteme pot fi sintetizate după cum urmează:

- protecția datelor împotriva unor prejudicii (*recuperarea – backup și recovery*);
- asigurarea accesului la date a unui număr mare de utilizatori (*concurența – concurrency*), precum și a mai multor aplicații;
- asigurarea integrității datelor, prin reguli stricte, dintre care cea mai cunoscută este *restricția de integritate referențială – referential integrity*;
- interzicerea accesului neautorizat la date (*securitatea – security*);
- izolarea detaliilor referitoare la manipularea datelor pe diferite platforme (*portabilitatea – portability*).

Recuperarea se referă la capacitatea SGBD-ului de a readuce baza de date într-o stare funcțională, în urma unor defecțiuni ale aplicațiilor ori sistemului de calcul ca atare. Dacă vom scrie o aplicație într-un limbaj de nivel înalt, dar stocarea și regăsirea datelor se va face prin acces direct la un sistem de fișiere, ne vom expune la un înalt risc de defecțiune la orice nivel. Va trebui să ținem cont de posibilitatea apariției acestei categorii de erori și de tratarea lor în aplicația concepută. Spre deosebire de limbajele de programare procedurale, ce au un caracter aproape universal, limbajele pentru baze de date implementează mecanismul *tranzacțiilor*. Tranzacția este o colecție de operații corelate, ce pot fi privite ca un tot unitar și tratate conform principiului „totul sau nimic". În multe cazuri, urmările unei erori software pot fi îndepărtate prin abandonarea tranzacției „vinovate". Acest lucru este posibil și datorită unui alt mecanism propriu SGBD, cel al *jurnalizării* tranzacțiilor.

Concurența este proprietatea bazei de date de a fi accesată de mai multe procese în același timp, fie acestea aplicații care se execută local sau la distanță. Deoarece concurența este strâns legată de recuperarea datelor, la baza ei stă tot mecanismul tranzacțiilor. Sistemele de gestiune a bazelor de date procedează la o „întrețesere" a tranzacțiilor efectuate de diferiți utilizatori, astfel încât aceștia nici nu sesizează accesul concurent. Această capacitate de întrețesere poartă numele de *serializabilitate*. În bazele de date relaționale, cele mai folosite (încă) în zilele noastre, serializarea tranzacțiilor se bazează pe blocarea (automată) a tabelelor și înregistrărilor accesate.

Independența datelor. Accesul concurent este un puternic motiv pentru ca bazele de date să necesite o organizare stratificată, în acord cu așa-zisele niveluri de abstractizare a datelor. Organizarea pe cele trei paliere: fizic sau intern, conceptual și extern permite, pe de o parte, modificarea modelelor de date fără a afecta aplicațiile existente, iar pe de altă parte, aplicații diferite pot fi realizate fără a fi nevoie de partajarea vreunei structuri conceptuale. Structurile de date dintr-o aplicație trebuie să reflecte nevoile aplicației respective, nu neapărat structurile de date memorate în bază. De aceea, prin comenzi adecvate datele se prelucrează în forma dorită a aplicației. Ar fi o strategie neînțeleaptă încărcarea pur și simplu a întregilor tabele în memorie, urmată de extragerea unor date din ele.

Restricția referențială este o facilitate a bazelor de date relaționale îndreptată spre menținerea coerenței datelor situate în diferite tabele. În bazele de date relaționale, această restricție se implementează prin valorile înseși ale datelor memorate, nu prin pointeri sau alte mecanisme.

Securitatea bazelor de date înseamnă protecția împotriva accesului sau modificărilor neautorizate (aici se consideră că accesul autorizat este întotdeauna sigur și nu afectează datele).

Prima și cea mai simplă măsură este izolarea fizică a sistemului de calcul de accesul persoanelor neautorizate (acolo unde este posibil).

Accesul la resursele sistemului poate avea loc prin facilități de natura *parolelor*, *profilelor utilizator* sau matrice ale drepturilor de acces (*privilegii*).

Portabilitatea software-ului este capacitatea acestuia de a se executa pe diferite platforme hardware. Proprietatea se regăsește la sistemele de operare (UNIX, Windows NT), interfețele bazate pe ferestre (X11, Windows) etc.

În cazul aplicațiilor cu baze de date, problema portabilității îmbracă două aspecte:

- portabilitatea la nivelul sistemului de gestiune a bazelor de date;
- portabilitatea la nivelul sistemului de operare.

Pentru respectarea portabilității la nivelul interfeței cu SGBD-ul, programatorul de aplicații trebuie să se limiteze la acele comenzi care sunt recunoscute de bazele de date avute în vedere. În cazul de față, el se va limita la standarde precum SQL2 sau (parțial) ODBC. Cât privește portabilitatea la nivelul sistemului de operare, fie doar aplicația, fie SGBD-ul, fie ambele trebuie să fie capabile să ruleze pe sistemele de operare vizate. ODBC poate fi o soluție și în acest caz. Iar dacă nu se poate asigura compatibilitatea la nivel de sistem de operare între aplicație și SGBD, tehnologia client-server constituie o alternativă elegantă. Se înțelege că, pentru a putea dezvolta o aplicație pentru diferite sisteme de operare, e necesar ca și instrumentele de dezvoltare a aplicației să fie compatibile la nivel de sisteme de operare.

Portarea unei aplicații necesită în majoritatea cazurilor recompilarea codului-sursă, ceea ce este totuși mai avantajos decât rescrierea aplicației pentru un anumit hardware ori sistem de operare.

Dacă suntem sinceri, trebuie să recunoaștem că totuși Visual FoxPro nu satisface pe deplin nici una din caracteristicile de mai sus. Astfel, în ceea ce privește recuperarea, cade în sarcina dezvoltatorului obligația construirii de la zero a unui mecanism de „backup și recovery", suportul nativ VFP în acest sens fiind nul. De asemenea, în ceea ce privește concurența, este adevărat că există în VFP anumite opțiuni foarte utile privind tranzacțiile și bufferingul în rețea, însă dacă veți testa soliditatea acestor mecanisme în condițiile, să

zicem, a 50 de utilizatori simultan veţi avea parte de surprize neplăcute, pentru că ceea ce pare „comis" definitiv la nivel VFP s-ar putea să nu fie astfel şi la nivelul serverului de fişiere (în fond, aplicaţiile VFP în reţea ar putea fi asimilate unei versiuni îmbunătăţite ale mecanismului de partajare a fişierelor). Apoi, în ceea ce priveşte restricţiile referenţiale observăm şi aici inexistenţa unui mecanism nativ de implementare a acestora, dezvoltatorii fiind nevoiţi să folosească mecanismul triggerelor; iar în ceea ce priveşte securitatea, ca şi în cazul recuperării, dezvoltatorii sunt nevoiţi să-şi construiască propriul sistem bazându-se eventual pe detalii ale sistemului de operare şi pe triggerele bazei de date. În fine, în ceea ce priveşte portabilitatea, din momentul în care FoxPro a devenit o marcă Microsoft, această caracteristică a dispărut (sau a fost uitată).

Serverele de baze de date sunt evident net superioare unui SGBD gen VFP. Care este însă dezavantajul acestora ? Răspunsul este simplu: preţul (considerând că cei care fac parte din tagma dezvoltatorilor sunt mereu dornici să înveţe lucruri noi, complexitatea unui astfel de mediu nedescurajându-i chiar de la început).

O dată cu creşterea spectaculoasă a dimensiunii BD în VFP, cu extinderea numărului de utilizatori ai bazei de date (presupunând, bineînţeles, că acestea sunt consecinţe naturale ale creşterii cifrei de afaceri, a veniturilor şi, prin urmare, şi a bugetului IT), apare drept imperios necesară trecerea la o arhitectură client-server pe două sau mai multe straturi. Cel mai la îndemână mod de extindere a aplicaţiilor VFP constă în păstrarea meniurilor, rapoartelor şi formularelor în VFP, trecerea bazei de date şi, implicit, a unei părţi importante din „logica aplicaţiei" într-un SGBD de tip server de baze de date. Firește, Microsoft recomandă migrarea spre SQL Server, însă serverul BD pe care poate fi „deplasată" baza de date Fox poate fi oricare dintre cele importante (Oracle, Sysbase, Informix, DB2), câtă vreme prin mecanismul ODBC conexiunea nu mai este o problemă.

16.2. Crearea schemei de baze de date pe serverul Oracle

Ne propunem în continuare să transformăm arhitectura aplicaţiei de până acum, bazându-ne pe un server de baze de date, şi anume Oracle8i, pentru definirea şi menţinerea schemei BD şi înregistrărilor aferente tabelelor, în partea de client mediul VFP trebuind să asigure interfaţa cu utilizatorul. Cu alte cuvinte, formularele şi rapoartele vor rămâne în VFP dar vor accesa şi actualiza datele rezidente pe serverul Oracle.

Un asemenea proces de migrare presupune parcurgerea în general a patru etape:

Etapa I: Portarea bazei de date în mediul serverului de baze de date, care ar trebui realizată în doi paşi: crearea schemei bazei de date (specificarea structurii tabelor şi declararea restricţiilor – inclusiv cele referenţiale) şi încărcarea datelor din mediul VFP în baza de date Oracle.

Etapa II: Configurarea mediului client/server sau, altfel spus, crearea canalului de legătură între clientul VFP şi serverul Oracle8i.

Etapa III: Crearea mecanismelor de acces şi actualizare (regăsire şi modificare/inserare/ştergere) a înregistrărilor din baza de date Oracle prin structurile de date specifice clientului VFP.

Etapa IV: Redefinirea formularelor şi rapoartelor ce vor rula pe clientul VFP, astfel încât să gestioneze corect datele rezidente pe serverul Oracle pe care le vor accesa prin mecanismele din etapa anterioară (inclusiv tratarea erorilor).

În această secţiune ne vom ocupa doar de prima etapă, celelalte constituind obiectul subcapitolelor următoare.

Migrarea bazei de date din VFP în Oracle s-ar putea realiza folosind un „vrăjitor" dedicat exclusiv unei asemenea sarcini, accesibil din meniul `Tools→Wizard→ Upsizing` prin selectarea din dialogul `Wizard Selection` a opţiunii `Oracle Upsizing Wizard`. Acesta ar trebui să refacă baza de date pe o platformă Oracle, redefinind structura tabelelor (ignorând însă detaliile legate de procedurile stocate, limbajul specific în Oracle fiind PL/SQL). Datorită faptului că n-am reuşit niciodată să finalizăm convingător această sarcină pe această cale (ne-am împiedicat întotdeauna de detalii „minore", cum ar fi conversia tipului de dată `DATE` al VFP în Oracle, detalii de care *wizard*-ul nu avea – sau poate nu dorea să aibă – cunoştinţă), dar şi pentru a avea un control mai strict asupra întregului proces, nu vom detalia funcţionalitatea instrumentului `Oracle Upsizing Wizard`.

În capitolele 2 şi 6 sunt propuse soluţiile şi scripturile de creare a bazei de date `Vinzari` în VFP. În esenţă, vom încerca să modificăm listingul următor adaptând comenzile SQL-DDL proprii VFP la canoanele comenzilor `DDL` specifice Oracle.

Listing 16.1. Scriptul de creare a bazei de date în VFP

```
CREATE DATABASE vinzari

CREATE TABLE judete ( ;
  jud CHAR(2) ;
    PRIMARY KEY ;
    CHECK (jud=LTRIM(UPPER(jud))) ;
        ERROR 'Indicativul judetului se scrie cu majuscule !', ;
  judet CHAR(25) ;
    UNIQUE ;
    NOT NULL ;
     CHECK (judet=LTRIM(PROPER(judet))) ;
                ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
                'denumirii judetului este majuscula; '+CHR(13)+;
                'restul literelor sunt mici!', ;
  regiune CHAR(15) ;
    DEFAULT 'Moldova' ;
    CHECK (INLIST(regiune, 'Banat', 'Transilvania', 'Dobrogea', 'Oltenia', 'Muntenia', 'Moldova')) ;
                ERROR 'Regiunea poate avea o valoare numai din lista:'+CHR(13)+;
                'Banat, Transilvania, Dobrogea, Oltenia, Muntenia, Moldova' ;
  )

CREATE TABLE localitati ( ;
  codpost CHAR(5) ;
    PRIMARY KEY ;
    CHECK (codpost=LTRIM(codpost)) ;
                ERROR 'Codul postal se scrie fara spatii la inceput !', ;
  loc CHAR(25) ;
    NOT NULL ;
    CHECK (loc=LTRIM(PROPER(loc))) ;
                ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
                'denumirii localitatii este majuscula; '+CHR(13)+;
                'restul literelor sunt mici!', ;
```

```
    jud CHAR(2) ;
      DEFAULT 'IS', ;
     FOREIGN KEY jud TAG jud REFERENCES judete TAG jud ;
    )

CREATE TABLE clienti ( ;
   codcl NUMBER(6) ;
      PRIMARY KEY  ;
      CHECK (codcl > 1000), ;
   dencl CHAR(30) ;
      CHECK (SUBSTR(dencl,1,1) = UPPER(SUBSTR(dencl,1,1))), ;
   codfiscal CHAR(9) ;
    NULL ;
      CHECK (SUBSTR(codfiscal,1,1) = UPPER(SUBSTR(codfiscal,1,1))), ;
   adresa CHAR(40) ;
    NULL ;
      CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))), ;
   codpost CHAR(5), ;
   telefon CHAR(10) ;
    NULL, ;
    FOREIGN KEY codpost TAG codpost REFERENCES localitati TAG codpost ;
   )

CREATE TABLE persoane ( ;
    cnp CHAR(14) ;
     PRIMARY KEY  ;
     CHECK (cnp=LTRIM(UPPER(cnp))) ;
               ERROR 'Codul numeric personal se scrie fara spatii la inceput !', ;
    nume CHAR(20) ;
     CHECK (nume=LTRIM(PROPER(nume))) ;
               ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
               'numelui este majuscula; '+CHR(13)+;
               'restul literelor sunt mici!', ;
    prenume CHAR(20) ;
     CHECK (prenume=LTRIM(PROPER(prenume))) ;
               ERROR 'Prima litera din fiecare cuvint al'+CHR(13)+;
               'prenumelui este majuscula; '+CHR(13)+;
               'restul literelor sunt mici!', ;
    adresa CHAR(40) ;
        NULL ;
        CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))) ;
               ERROR 'Prima litera din adresa este obligatoriu majuscula !', ;
   sex CHAR(1) DEFAULT 'B' ;
     CHECK (INLIST(sex,'F','B')) ;
               ERROR 'Atributul Sex poate avea valorile F (de la Femeiesc)'+CHR(13)+;
                     'sau B (de la Barbatesc) !', ;
   codpost CHAR(5), ;
   telacasa CHAR(10) NULL, ;
   telbirou CHAR(10) NULL, ;
   telmobil CHAR(10) NULL, ;
   email CHAR(20) NULL, ;
    FOREIGN KEY codpost TAG codpost REFERENCES localitati TAG codpost ;
   )

CREATE TABLE persclienti ( ;
   cnp CHAR(14), ;
   codcl NUMBER(6), ;
   functie CHAR(25) ;
     CHECK (SUBSTR(functie,1,1) = UPPER(SUBSTR(functie,1,1))) ;
```



```
               ERROR 'Prima litera din functie este obligatoriu majuscula !', ;
    PRIMARY KEY cnp+STR(codcl,6)+functie TAG primaru, ;
     FOREIGN KEY cnp TAG cnp REFERENCES persoane TAG cnp, ;
     FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl ;
    )

CREATE TABLE produse ( ;
   codpr NUMBER(6) ;
     PRIMARY KEY ;
     CHECK (codpr > 0) ;
               ERROR 'Codul produsului trebuie sa fie mai mare de 100000 !', ;
   denpr CHAR(30) ;
     CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))) ;
               ERROR 'Prima litera din denumirea produsului este obligatoriu majuscula !', ;
   um CHAR(10), ;
   grupa CHAR(15) ;
     CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))) ;
               ERROR 'Prima litera din grupa este obligatoriu majuscula !', ;
   procTVA NUMBER(3,2) ;
    DEFAULT .22 ;
   )

CREATE TABLE facturi ( ;
   nrfact NUMBER(8) ;
     PRIMARY KEY, ;
   datafact DATE ;
    DEFAULT DATE() ;
       CHECK (BETWEEN(datafact,{01/08/2000},{31/12/2010})) ;
               ERROR 'Baza de date functioneaza in intervalul 1 aug.2000 - 31 dec.2010 !', ;
   codcl NUMBER(6), ;
   Obs CHAR(50) NULL, ;
   valftva NUMBER(16) DEFAULT 0 CHECK (ValFTVA < 100000000) ;
    ERROR 'Valoarea fara TVA a unei facturi nu poate depasi 100 000 000 (lei) !'  , ;
   FOREIGN KEY codcl TAG codcl REFERENCES clienti TAG codcl ;
   )

CREATE TABLE liniifact ( ;
   nrfact NUMBER(8), ;
   linie NUMBER(2) ;
       CHECK (linie > 0) ;
               ERROR 'Atributul linie trebuie sa fie mai mare ca zero !', ;
   codpr NUMBER(6), ;
   cantitate NUMBER(10), ;
   pretunit NUMBER (12), ;
   PRIMARY KEY STR(nrfact,8)+STR(linie,2) TAG primaru, ;
    FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact, ;
    FOREIGN KEY codpr TAG codpr REFERENCES produse TAG codpr ;
   )

CREATE TABLE incasari ( ;
   codinc NUMBER(8) ;
       PRIMARY KEY, ;
   datainc DATE ;
    DEFAULT DATE() ;
     CHECK (BETWEEN(datainc,{01/08/2000},{31/12/2010})) ;
               ERROR 'Baza de date functioneaza in intervalul 1 aug.2000 - 31 dec.2010 !', ;
   coddoc CHAR(4) ;
       CHECK(coddoc=UPPER(LTRIM(coddoc))) ;
               ERROR 'Codul documentului se scrie cu majuscule !', ;
```

```
    nrdoc CHAR(16), ;
    datadoc DATE ;
      DEFAULT DATE() - 7 ;
      CHECK (BETWEEN(datadoc, {01/01/2000},{31/12/2010})) ;
                ERROR 'Data documentului trebuie sa fie intre 1 ian.2000 si 31 dec.2010 !' ;
    )

  CREATE TABLE incasfact ( ;
    codinc NUMBER(8), ;
      nrfact NUMBER(8), ;
    transa NUMBER(16) ;
      NOT NULL, ;
    PRIMARY KEY STR(codinc,8)+STR(nrfact,8) TAG primaru, ;
      FOREIGN KEY codinc TAG codinc REFERENCES incasari TAG codinc, ;
      FOREIGN KEY nrfact TAG nrfact REFERENCES facturi TAG nrfact ;
    )
```

Vom converti acest script ţinând cont de următoarele caracteristici ale limbajului SQL-DDL propriu Oracle:

- în primul rând, frazele SQL din Oracle se pot întinde pe mai multe rânduri (nedespărţite prin caractere speciale cum este „;" în VFP) finalul lor fiind însă declarat prin caracterul „;" ;
- în fraza `CREATE TABLE` din VFP, restricţiile de la nivelul câmpurilor sunt despărţite prin caracterul „,", iar câmpurile – prin lipsa oricărui caracter special (deci pur şi simplu prin spaţiu). În mediul Oracle însă, restricţiile la nivelul câmpurilor nu sunt despărţite prin caracterul „,", prezenţa acestuia semnificând încheierea declaraţiilor pentru un câmp (adică nume + tip dată + restricţii) sau încheierea declaraţiilor pentru fiecare restricţie de la nivelul întregii tabele (nu la nivelul unui câmp) precizate în finalul frazei `CREATE TABLE`;
- în ceea ce priveşte tipurile de date, vom respecta următoarele reguli:
  - tipul VFP `Number` va corespunde cu `NUMBER` şi în Oracle;
  - tipul `Char` va corespunde în Oracle cu `CHAR` dacă respectivul câmp face parte dintr-o cheie (primară sau străină), sau cu `VARCHAR2`[71] în majoritatea celorlalte cazuri, excepţie făcând situaţiile în care respectivele câmpuri au în mod obligatoriu o lungime fixă;
  - pentru tipul `Logic` nu există în Oracle un tip corespunzător, aşa încât acesta poate fi convertit fie în `CHAR`, fie în `NUMBER`, iar respectivului câmp îi va fi declarat un `check` care desemna un domeniu de două valori corespunzătoare poziţiilor `.T.` (Adevărat) şi `.F.` (Fals) din VFP;
  - tipurile `Date` şi `DateTime` vor fi convertite în tipul `DATE` al Oracle.
- în ceea ce priveşte restricţiile din VFP, ele corespund în mare celor permise şi în Oracle (VFP `Primary Key` – Oracle `PRIMARY KEY`, VFP `Check` – Oracle `CHECK`, VFP `Not null` – Oracle `NOT NULL`, VFP `Foreign Key` – Oracle `FOREIGN KEY`, VFP `Unique` – Oracle `UNIQUE KEY`). Va trebui să ţinem însă seama de următoarele detalii:

71. Tipul `VARCHAR2` se recomandă pentru cazurile în care dimensiunea câmpului este variabilă, declarându-se astfel doar lungimea maximă. În acest fel se obţine o economie de spaţiu, valorilor din respectiva coloană fiindu-le rezervat spaţiu fizic numai pentru lungimea lor reală.

- funcţia `Proper` din VFP (invocată în `Check`-uri) este echivalentă cu funcţia `INITCAP` din Oracle;
- funcţia `InList()` din VFP va fi transformată în Oracle într-o expresie conţinând operatorul `IN`;
- funcţia `Date()` din VFP (prin care se obţine data sistemului) este echivalentă cu funcţia `SYSDATE` din Oracle;
- funcţia `Between()` din VFP va fi echivalată cu operatorul `BETWEEN` din Oracle, care are următoarea sintaxă:
- *<Expresie evaluată>* `BETWEEN` *<expresie limită inferioară>* `AND` *<Expresie limită superioară>*;
- invocarea unei valori tip dată calendaristică în VFP (prin expresii de genul `{01/01/2002}`) va fi tradusă în Oracle prin intermediul funcţiei `TO_DATE`[72];
- în restricţiile `PRIMARY KEY`, `UNIQUE KEY` şi `FOREIGN KEY` din Oracle sunt specificate câmpurile (şi eventual tabelele) participante şi nu expresii de indexare ca în VFP, construirea şi gestionarea expresiilor indecşilor corespunzători nefiind transparentă dezvoltatorilor. De exemplu, expresia VFP:
- `PRIMARY KEY cnp+STR(codcl,6)+functie TAG primaru`
- va corespunde în Oracle cu:
- `CONSTRAINT Primaru PRIMARY KEY (cnp, codcl, functie).`

În funcţie de specificul bazei de date VFP ce urmează a fi convertită, apar diferite astfel de situaţii particulare; de aceea, pentru rezolvarea corectă a acestora, este necesară o consultare detaliată a documentaţiei Oracle privind sintaxa comenzilor DDL, tipurile de date disponibile, sintaxa operatorilor SQL specifici, sintaxa funcţiilor folosite în declaraţii de tip `CHECK` etc.

În fine, ţinând cont de precizările de mai sus, vom ajunge la un script DDL în Oracle care, într-o formă validă, va arăta în linii mari aşa cum este prezentat în listingul 16.2.

Listing 16.2. Listing pentru crearea bazei de date Oracle

```
DROP TABLE INCASFACT;
DROP TABLE INCASARI;
DROP TABLE LINIIFACT;
DROP TABLE FACTURI;
DROP TABLE PRODUSE;
DROP TABLE PERSCLIENTI;
DROP TABLE PERSOANE;
DROP TABLE CLIENTI;
DROP TABLE LOCALITATI;
DROP TABLE JUDETE;
```

72. În Oracle, o valoare tip dată calendaristică (`DATE`) nu poate fi scrisă într-o expresie printr-o sintaxă specifică, motiv pentru care se foloseşte funcţia `TO_DATE(<expresie dată calendaristică - şir de caractere>, <format interpretare - şir de caractere>)`.

```
CREATE TABLE judete (
   jud CHAR(2)
     PRIMARY KEY
     CHECK (jud=LTRIM(UPPER(jud))),
   judet VARCHAR2(25)
     NOT NULL
     UNIQUE
      CHECK (judet=LTRIM(INITCAP(judet))),
   regiune VARCHAR2(15)
     DEFAULT 'Moldova'
     CHECK (Regiune IN ('Banat', 'Transilvania','Dobrogea','Oltenia',
      'Muntenia','Moldova'))
    )
/

CREATE TABLE localitati (
   codpost CHAR(5)
      PRIMARY KEY
      CHECK (codpost=LTRIM(codpost)),
   loc VARCHAR2(25)
      NOT NULL
      CHECK (loc=LTRIM(INITCAP(loc))),
   jud CHAR(2)
      DEFAULT 'IS'
     REFERENCES judete(jud)
   )
/

CREATE TABLE clienti (
   codcl NUMBER(6)
      PRIMARY KEY
      CHECK (codcl > 1000),
   dencl VARCHAR2(30)
      CHECK (SUBSTR(dencl,1,1) = UPPER(SUBSTR(dencl,1,1))),
   codfiscal CHAR(9)
      CHECK (SUBSTR(codfiscal,1,1) = UPPER(SUBSTR(codfiscal,1,1))),
   adresa VARCHAR2(40)
      CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))),
   codpost CHAR(5) REFERENCES localitati(codpost),
   telefon VARCHAR2(10)
   )
/

CREATE TABLE persoane (
    cnp CHAR(14)
     CONSTRAINT pk_persoane PRIMARY KEY
     CONSTRAINT ck_cnp CHECK (cnp=LTRIM(UPPER(cnp))),
    nume VARCHAR2(20)
      CONSTRAINT ck_nume  CHECK (nume=LTRIM(INITCAP(nume))),
    prenume VARCHAR2(20),
      CONSTRAINT ck_prenume CHECK (prenume=LTRIM(INITCAP(prenume))),
    adresa VARCHAR2(40)
      CONSTRAINT ck_adresa CHECK (SUBSTR(adresa,1,1) = UPPER(SUBSTR(adresa,1,1))),
    sex CHAR(1) DEFAULT 'B'
       CONSTRAINT ck_sex CHECK (sex = 'F' OR SEX = 'B'),
    codpost CHAR(5) CONSTRAINT fk_pers_loc REFERENCES localitati(codpost),
    telacasa VARCHAR2(10) NULL,
    telbirou VARCHAR2(10) NULL,
    telmobil VARCHAR2(10) NULL,
```



```
    email VARCHAR2(20) NULL
   )
/

CREATE TABLE persclienti (
   cnp CHAR(14),
   codcl NUMBER(6),
   functie VARCHAR2(25)
      CHECK (SUBSTR(functie,1,1) = UPPER(SUBSTR(functie,1,1))),
   CONSTRAINT pk_persclieni PRIMARY KEY (cnp, codcl, functie),
   CONSTRAINT fk_persoane_clienti FOREIGN KEY (cnp) REFERENCES persoane(cnp),
   CONSTRAINT fk_clienti_persclienti FOREIGN KEY (codcl) REFERENCES clienti(codcl)
   )
/

CREATE TABLE produse (
   codpr NUMBER(6)
      PRIMARY KEY
      CHECK (codpr > 0),
   denpr VARCHAR2(30)
      CHECK (SUBSTR(denpr,1,1) = UPPER(SUBSTR(denpr,1,1))),
   um VARCHAR2(10),
   grupa VARCHAR2(15)
      CHECK (SUBSTR(grupa,1,1) = UPPER(SUBSTR(grupa,1,1))),
   procTVA NUMBER(3,2)
    DEFAULT .22
   )
/

CREATE TABLE facturi (
   nrfact NUMBER(8)
      PRIMARY KEY,
   datafact DATE
     DEFAULT SYSDATE
     CHECK (Datafact BETWEEN TO_DATE('01/08/2000', 'DD/MM/YYYY')
                AND TO_DATE('31/12/2010', 'DD/MM/YYYY')),
   codcl NUMBER(6),
   Obs VARCHAR2(50) NULL,
   valftva NUMBER(16) DEFAULT 0 CHECK (ValFTVA < 100000000),
   CONSTRAINT fk_clienti_facturi FOREIGN KEY (codcl) REFERENCES clienti(codcl)
   )
/

CREATE TABLE liniifact (
   nrfact NUMBER(8),
   linie NUMBER(2)
     CHECK (linie > 0),
   codpr NUMBER(6),
   cantitate NUMBER(10),
   pretunit NUMBER (12),
   CONSTRAINT pk_liniifact PRIMARY KEY (nrfact,linie),
   CONSTRAINT fk_facturi_liniifact FOREIGN KEY (nrfact) REFERENCES facturi(nrfact),
   CONSTRAINT fk_produse_liniifact FOREIGN KEY (codpr) REFERENCES produse(codpr)
   )
/

CREATE TABLE incasari (
   codinc NUMBER(8) PRIMARY KEY,
   datainc DATE
```

```
    DEFAULT SYSDATE
     CHECK (datainc BETWEEN TO_DATE('01/08/2000', 'DD/MM/YYYY')
              AND TO_DATE('31/12/2010', 'DD/MM/YYYY')),
   coddoc VARCHAR2(4)
    CHECK(coddoc=UPPER(LTRIM(coddoc))),
   nrdoc VARCHAR2(16),
   datadoc DATE
    DEFAULT SYSDATE - 7
     CHECK (datadoc BETWEEN TO_DATE('01/08/2000', 'DD/MM/YYYY')
              AND TO_DATE('31/12/2010', 'DD/MM/YYYY'))
  )
/

CREATE TABLE incasfact (
  codinc NUMBER(8),
   nrfact NUMBER(8),
  transa NUMBER(16) NOT NULL,
  CONSTRAINT pk_incasfact PRIMARY KEY (codinc,nrfact),
  CONSTRAINT fk_incasari_incasfact FOREIGN KEY (codinc) REFERENCES incasari(codinc),
  CONSTRAINT fk_facturi_invcasfact FOREIGN KEY (nrfact) REFERENCES facturi(nrfact)
  )
/
```

Acest script poate fi rulat folosind utilitarul `Oracle SQL*Plus`, de aceea execuţia fiecărei comenzi este invocată prin caracterul „/" (echivalent este şi caracterul „;", care ar putea să însoţească ultimul caracter „) " al unei comenzi `CREATE TABLE`).

În prealabil, presupunând că pe server a fost deja creată o bază de date Oracle accesibilă prin stringul `VINZARI`, ar trebui executate (tot din `Oracle SQL*Plus`) următoarele comenzi:

- pentru crearea schemei care va găzdui tabele din listingul de mai sus:

```
CREATE USER vinzari IDENTIFIED BY vinzari
DEFAULT TABLESPACE users,
TEMPORARY TABLESPACE temp
/ [73]
GRANT connect, resource TO vinzari
/
```

- pentru executarea scriptului din listingul 16.2 (presupunem că este salvat în directorul `Migrare_Oracle\Scripturi` al locaţiei aplicaţiei):

```
CONNECT vinzari/vinzari@vinzari
@K:\Aplicatie\Migrare_Oracle\Creare_BD_Oracle.sql
```

Verificăm existenţa tabelelor prin următoarea frază:

```
select table_name from user_tables;
```

şi va trebui să obţinem următorul rezultat:

```
TABLE_NAME
------------------------------
CLIENTI
FACTURI
INCASARI
```

73. Detaliu din punctul de vedere al administrării BD: este recomandabilă crearea unui *tablespace* special pentru schema `vinzari`.


```
INCASFACT
JUDETE
LINIIFACT
LOCALITATI
PERSCLIENTI
PERSOANE
PRODUSE

10 rows selected.
```

În pasul doi al acestei prime etape din procesul de migrare, ne-am propus să populăm tabelele create în baza de date Oracle. Vom încerca să realizăm acest lucru în felul următor: pentru fiecare tabelă din baza de date VFP vom genera un script de populare (fişier text cu extensia `.SQL` executabil folosind utilitarul `Oracle SQL*Plus`) care va conţine frazele `INSERT-SQL` necesare (bineînţeles, respectând sintaxa specifică Oracle), apoi vom centraliza apelurile de execuţie ale acestor scripturi într-un alt fişier cu extensia `.SQL` asociat bazei de date.

Pentru generarea fişierelor de comenzi mai sus amintite, interpretabile prin `Oracle SQL*Plus`, vom scrie o procedură în VFP care să prelucreze datele din baza de date sursă. Problemele mai serioase ridicate de crearea acestei proceduri se referă la scrierea în şirul care formează fraza `SQL-INSERT` a expresiilor pentru valorile corespunzătoare câmpurilor respectând sintaxa specificaţiilor limbajului SQL din Oracle8i. În acest scop, în listingul 16.3 veţi găsi definită o funcţie specială `format_expresie_valoare()` care rezolvă această problemă pentru tipurile de date întâlnite în baza de date VFP considerată ca exemplu (vezi listingul 16.1). După cum se poate uşor observa, listingul 16.3 este format din trei părţi:

- prima parte conţine programul de bază care va crea fişierul ce invocă procedura responsabilă de generarea scripturilor conţinând comenzile `INSERT-SQL` pentru fiecare tabelă a bazei de date. Pentru a afla care sunt tabelele bazei de date, a fost utilizată funcţia `ADBOBJECTS()` ce va plasa într-un masiv numele acestora;
- a doua parte cuprinde programul prin care sunt create fişierele de populare ale fiecărei tabele din baza de date. Aici se va construi de fapt câte un şir ce va conţine fraza `INSERT` corespunzătoare fiecărei înregistrări din tabela-sursă. Clauza `VALUES` a frazelor `INSERT-SQL` va fi formată ţinând cont de sintaxa specifică precizării valorilor din Oracle. Acest lucru este condiţionat de tipul de dată, motiv pentru care am creat funcţia `format_expresie_valoare()` ce realizează o conversie a tipurilor din VFP în Oracle;
- a treia parte o constituie funcţia `format_expresie_valoare()`. Aceasta realizează o conversie a valorilor din VFP în Oracle pentru tipurile de dată: N-Numeric, C-Character, D-Date, L-Logic, T-DateTime[74].

74. Ne-am limitat doar la tipurile de date VFP de care ne-am lovit la crearea schemei în Oracle pe baza tabelelor VFP din listingul 16.1. Dacă în baza VFP iniţială ar mai fi implicate şi alte tipuri de date, funcţia `format_expresie_valoare()` va trebui completată pentru a trata şi aceste cazuri.

Listing 16.3

```
*** Obtinere script pentru populare bazei de date de pe serverul Oracle
PROCEDURE Generare_Script_Pop_BD_Oracle
PARAMETER numeBazaDate, DirScriptGenerat
Local nrtabele, a_tabele(1), nume_script_pop_tbl, numescriptgenerat
IF !dbused(numeBazaDate)
  OPEN DATABASE (NumeBazaDate) SHARED
ENDIF
IF EMPTY(dirscriptgenerat)
   DirScriptGenerat = SYS(5)+SYS(2003)
ENDIF
numeScriptGenerat=DirScriptGenerat+'\populare_bd_'+numeBazaDate+'.sql'
MESSAGEBOX('Atentie! Veti obtine un script localizat in '+numeScriptGenerat+' !')

nrtabele = ADBOBJECTS(a_tabele, "TABLE")

** Obtin mai intai scriptul general din care vor fi invocate
** scripturile pentru populatea fiecarei tabele
SET TEXTMERGE ON
SET TEXTMERGE TO (numeScriptGenerat) NOSHOW
\--Script populare BAZA DE DATE
\\<<numeBazaDate>>
\--Generat la
\\<<DATETIME()>>
FOR i=1 TO ALEN(a_tabele, 1)
  IF EMPTY(numescriptgenerat)
    nume_script_pop_tbl = SYS(5)+SYS(2003)+'\populare_'+a_tabele(i)+'.sql'
  ELSE
    nume_script_pop_tbl = DirScriptGenerat+'\populare_'+a_tabele(i)+'.sql'
  ENDIF
  \@@
  \\<<nume_script_pop_tbl>>
ENDFOR
\
\COMMIT;
SET TEXTMERGE TO
SET TEXTMERGE OFF

** Invoc procedura Generare_Insert_SQL pentru a obtine
** cate un script de populare separat pentru fiecare tabela din baza de date
FOR j=1 TO ALEN(a_tabele, 1)
**  messagebox(DirScriptGenerat+a_tabele(j))
  Generare_Insert_SQL(a_tabele(j), DirScriptGenerat)
ENDFOR
ENDPROC && Sfarsit procedura principala Generare_Script_Pop_BD_Oracle

*-------------------------------------------------------------------------------------
PROCEDURE Generare_Insert_SQL
*** Generarea frazelor INSERT-SQL corespunzatoare inregistrarilor unei tabele FOX
PARAMETER numetabela, DirScript
Local nrcampuri, a_campuri(1), sir_clauza_campuri, sir_clauza_values, sir_sql_cmd
CLEAR
SET STRICTDATE TO 0
SET DATE TO BRITISH
SET CENTURY ON
IF !USED(numetabela)
  use (numetabela) in 0 shared
ENDIF
```



```
  nrcampuri = afield(a_campuri, numetabela)
  ** col 1 - numele campului
  ** col 2 - tipul campului N-Number, C-Character, D-Date, T-DateTime, L-Logic
  ** col 3 - dimesiunea campului
  ** col 4 - dimensiunea partii zecimale
  ** In locul functie afield se pot folosi functiile:
  ** FIELD(<nr camp>) - pentru a gasi numele campului
  ** TYPE(FIELD(<nrcamp>)) - pentru a afla tipul campului
  select (numetabela)
  sir_clauza_campuri=''
  sir_clauza_values=''
  sqlcmd=''
  IF EMPTY(DirScript)
  **   numescript='K:\Aplicatie\Migrare_Oracle\populare_'+numetabela+'.sql'
       numescript=SYS(5)+SYS(2003)+'\populare_'+numetabela+'.sql'
  ELSE
     numescript=DirScript+'\populare_tbl_'+numetabela+'.sql'
  ENDIF
  ? 'Atentie! Veti obtine un script localizat in '+numescript+' !'

  FOR i=1 TO ALEN(a_campuri, 1) && sau FCOUNT()
      IF INLIST(a_campuri(i, 2), 'N','C','D','T', 'L') && sau INLIST(TYPE(FIELD(i)), 'N','C','D','T', 'L')
          sir_clauza_campuri=sir_clauza_campuri+a_campuri(i, 2) && sau +FIELD(i)
          IF i<>ALEN(a_campuri, 1)  && sau FCOUNT()
                  sir_clauza_campuri=sir_clauza_campuri+","
          ENDIF
      ENDIF
  ENDFOR
  ** Directionez textul generat in fisierul script declarat anterior
  SET TEXTMERGE ON
  SET TEXTMERGE TO (numescript) NOSHOW
  \--Script populare tabela
  \\<<numetabela>>
  \--Generat la
  \\<<DATETIME()>>
  nrinreg=RECCOUNT()
  SCAN
      FOR i=1 TO ALEN(a_campuri, 1) && sau FCOUNT()
          IF INLIST(a_campuri(i, 2), 'N','C','D','T', 'L') && sau INLIST(TYPE(FIELD(i)), 'N','C','D','T', 'L')
          && Formatul sintactic al expresiei prin care se precizeaza valoarea campului in fraza INSERT
          && depinde de tipul campului - vezi mai jos functia format_expresie_valoare()
                  sir_clauza_values=sir_clauza_values+format_expresie_valoare(a_campuri(i,   1))
&& +delimitator(FIELD(i))
                  IF i<>ALEN(a_campuri, 1)  && sau FCOUNT()
                          sir_clauza_values=sir_clauza_values+","
                  ENDIF
          ENDIF
      ENDFOR
      sqlcmd='INSERT INTO '+numetabela+' VALUES ('+sir_clauza_values+');'
      WAIT WIND 'Asteptati <<< '+ ALLT(STR(RECNO()))+'/'+ALLT(STR(nrinreg)) NOWAIT
      \<<sqlcmd>>
      sir_clauza_values=''
  ENDSCAN
  \
  \COMMIT;
  SET TEXTMERGE TO
  SET TEXTMERGE OFF
  WAIT WIND "Gata!" NOWAIT
  ENDPROC && Sfarsit procedura Generare_Insert_SQL
```

```
*-----------------------------
FUNCTION format_expresie_valoare
PARAMETERS numevar
DO CASE
    case ISNULL(&numevar)
        RETURN 'NULL'
    CASE VARTYPE(&numevar)='N'
        RETURN ALLT(STR(&numevar,16,2))
    CASE VARTYPE(&numevar)='C'
        RETURN CHR(39)+ALLT(&numevar)+CHR(39)
    CASE VARTYPE(&numevar)='D' OR VARTYPE(&numevar)='T'
        RETURN
'TO_DATE('+CHR(39)+DTOC(&numevar)+CHR(39)+','+CHR(39)+'DD/MM/YYYY
HH:MI:SS'+CHR(39)+')'
    CASE VARTYPE(&numevar)='L'
        RETURN allt(str(IIF(&numevar,1,0)))
    OTHERWISE
        RETURN '<unknown>'
ENDCASE
ENDFUNC
```

În urma execuției acestei proceduri poate fi obținut un fișier cum este cel din listingul 16.4 și totodată scripturile invocate în acest fișier (vezi figura 16.1) .

Atenție! După generarea scriptului pentru popularea bazei de date din Oracle, verificați succesiunea apelurilor către fișierele de populare ale tabelelor individuale. Pentru a nu avea probleme cu restricțiile referențiale, mai întâi trebuie invocate scripturile corespunzătoare tabelelor-părinte și apoi cele ale tabelelor-copil.

Listing 16.4

```
--Script populare BAZA DE DATEVINZARI
--Generat la  29/11/2001 06:30:27 PM
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_JUDETE.sql
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_LOCALITATI.sql
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_CLIENTI.sql
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_PERSOANE.sql
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_PERSCLIENTI.sql
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_PRODUSE.sql
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_FACTURI.sql
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_LINIIFACT.sql
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_INCASARI.sql
@@K:\APLICATIE\MIGRARE_ORACLE\Scripturi\populare_INCASFACT.sql
COMMIT;
```

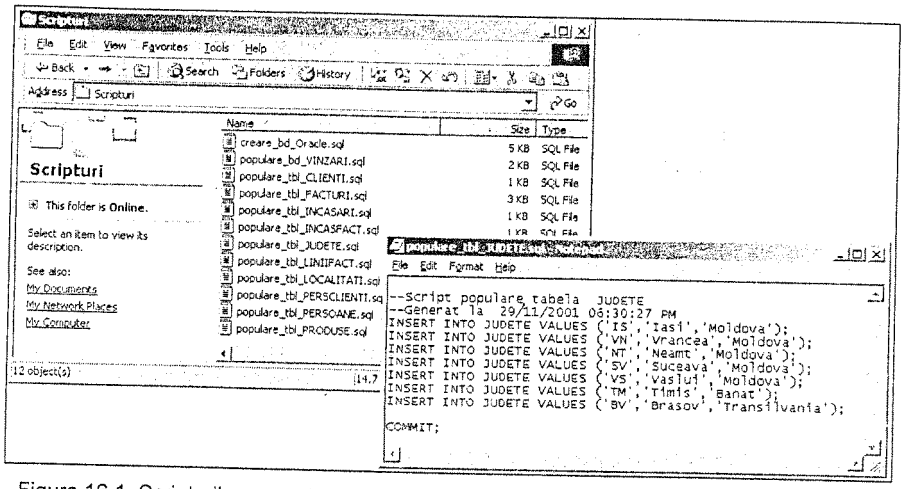



Figura 16.1. Scripturile generate și conținutul fișierului `populare_tbl_judete.sql`

Singurul lucru care mai rămâne de făcut ar fi rularea scriptului `Populare_BD_Vinzari.SQL` din `Oracle SQL*Plus`.

16.3. Configurarea mediului client-server: client VFP – server Oracle8i

16.3.1. ODBC, drivere ODBC

Open Database Connectivity (ODBC) reprezintă o interfață standard care permite unei aplicații să acceseze date din surse diferite: Oracle, MS SQL Server, DB2, Informix, MS Access, dBase ș.a. Inițiativa ODBC a aparținut companiei Microsoft, având acceptul și concursul firmelor de soft care produc servere BD.

Legătura între aplicațiile ce exploatează aceste surse de date și serverele BD se realizează prin *drivere* specifice, care reprezintă, în fapt, DLL-uri invocate atunci când se solicită accesul la o anumită sursă de date. Prin urmare, o aplicație se poate conecta la oricare sursă pentru care există drivere (vezi figura 16.2).

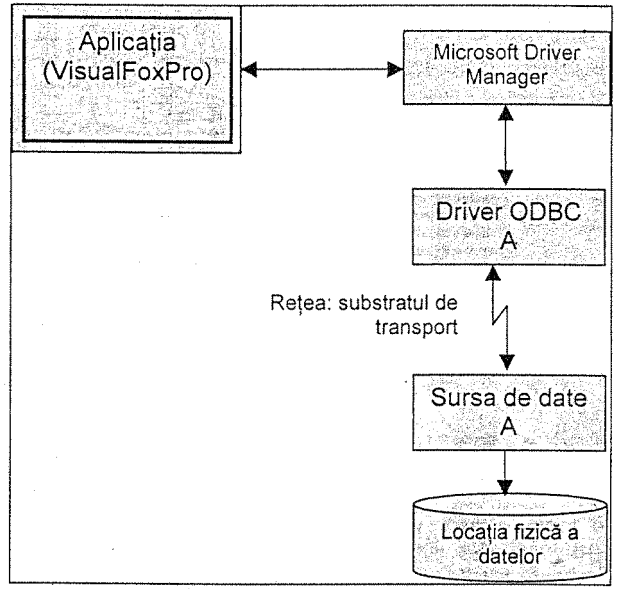


Figura 16.2. Componentele ODBC

Conform intențiilor inițiatorilor, interfața ODBC definește:

- o bibliotecă de apeluri de funcții care permit unei aplicații să se conecteze la o sursă de date, să execute fraze SQL și să primească rezultatele acestora;
- modalitate standard de conectare la o sursă (BD) de date externă;
- modalitate standard de reprezentare a tipurilor de date.

Mai trebuie menționat că această interfață mai permite și recepționarea, tot printr-un mecanism standardizat, a erorilor generate de motoarele diferitelor servere de baze de date care gestionează BD sursă.

Sintaxa frazelor SQL prin care sunt apelabile sursele de date este (sau ar trebui să fie) conformă cu specificațiile standardului SQL-92. Prin urmare, sarcina traducerii specificațiilor SQL-92 în specificațiile proprii serverelor ce gestionează diferitele surse de date aparține tot driverelor ODBC.

16.3.2. Clientul ORACLE8i și protocolul de conectare NET8

Într-o configurație client-server, dialogul dintre cele două entități poate fi rezumat astfel: când un client trimite o cerere (frază SQL) către un server BD, acesta o recepționează, o execută, după care trimite ca răspuns rezultatul frazei SQL sau erorile care rezultă în urma execuției. O astfel de comunicare simplă și comodă între clienți și serverele de baze de date este posibilă în tehnologia Oracle prin protocolul NET8.

Protocolul NET8 se sprijină pe substratul de transport din stiva protocoalelor de rețea (TCP/IP, SPX/IPX). NET8 furnizează în principal trei funcții de bază care se referă la:

- operațiile de conectare ;
- operațiile privitoare la date (transferul datelor între client și server);
- operațiile legate de excepțiile/evenimentele anormale ce pot surveni în cadrul unei sesiuni client-server.

O sesiune de lucru Oracle nu poate fi descrisă fără a avea o imagine, chiar și sumară, a ceea ce se întâmplă cu cererile clienților o dată ajunse pe mașina pe care rulează serverul Oracle8i. Rolul esențial în această privință îl are un serviciu special (în NT sau 2000) numit `LISTENER`. Acesta recepționează cererile trimise către serverul Oracle prin diferitele protocoale de rețea și le redirecționează (rutează) către bazele de date (sau, mai specific, către instanțele bazelor de date) cărora le sunt adresate efectiv.

Procedura de conectare a unui client la un server Oracle se reduce în principal la următorii pași:

1. Un program sau aplicație-utilizator inițiază o cerere de conectare trimițând numele de utilizator, parola și un nume de serviciu (`service name`), care poate fi asimilat unui alias al bazei de date destinație. Acest `service name` este mapat direct pe un descriptor de conectare (`connect descriptor`). Maparea cade exclusiv în sarcina protocolului NET8, care va încerca să o rezolve citind fișierele de configurare în care se găsesc specificațiile descriptorilor de conectare și metoda de rezoluție.

2. După ce semnificația numelui de serviciu este determinată, cererea de conectare este transmisă, prin protocolul de rețea existent, serviciului `LISTENER` de pe mașina server a cărui adresă a fost dedusă din descriptorul de conectare obținut prin operația de mapare descrisă mai sus.

3. Sesiunea de lucru dintre client și serverul bazei de date nu este însă susținută de către `LISTENER`, rolul acestuia limitându-se doar la redirecționarea cererilor către instanțele bazelor de date cărora le sunt adresate. Astfel, când primește o cerere către o bază de date, `LISTENER`-ul creează un proces server căruia îi predă controlul sesiunii deschise între server și client.

4. Adresa procesului nou-creat este trimisă procesului de pe mașina clientului. Astfel, cele două procese (procesul-utilizator – care a inițiat cererea de conectare – și procesul-server) vor comunica direct pe durata sesiunii fără a implica serviciul `LISTENER`.

16.3.3. Arhitectura funcțională a driverului Oracle ODBC Driver

Oracle ODBC Driver permite aplicațiilor Windows (95, 98, NT sau 2000) să efectueze operații de scriere (actualizare) sau citire (interogare) în BD Oracle utilizând software-ul de comunicare proprietar Oracle, NET8. Se asigură, astfel, independența față de protocolul de transport (TCP/IP, IPX/SPX ș.a.) a dialogului dintre clienții și serverele ORACLE.

Acest driver folosește interfața OCI (*Oracle Call Interface*) de pe client pentru a trimite cererile de acces ale aplicației și pentru a recepționa rezultatele acestora de la sursa de date invocată. Protocolul de comunicare NET8 este utilizat pentru realizarea legăturilor între clientul OCI și serverul Oracle.

Driverul furnizat de Oracle traduce sintaxa SQL proprie interfeței ODBC în sintaxa proprie serverului sursei de date. În momentul în care sursa de date invocată returnează rezultatul, același driver îl traduce în sintaxa SQL a ODBC.

Arhitectura funcțională a mecanismului de comunicare prin Oracle ODBC driver este prezentată în figura 16.3.

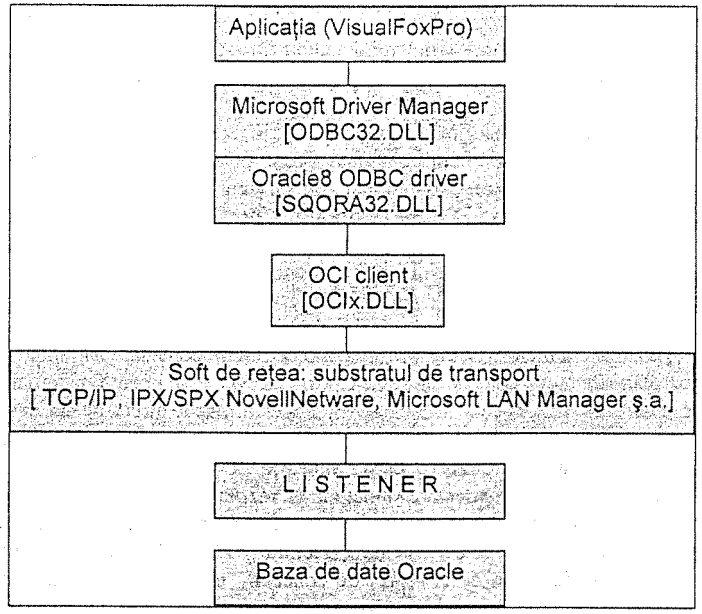


Figura 16.3. Mecanismul de comunicare prin Oracle ODBC driver

Se observă că peste stratul de transport al softului de rețea se suprapun componente ale protocolului NET8 pentru care au fost construite adaptoare aferente tuturor protocoalelor de rețea importante.

Dintre componentele arhitecturale ale protocolului NET8, interfața OCI și serviciul `LISTENER` sunt prezentate în schema de mai sus.

OCI constituie primul strat din stiva NET8, iar codul OCI (adică funcțiile din biblioteca `OCI.dll`) conține toate informațiile necesare pentru inițierea și desfășurarea dialogului client-server ORACLE – sau, cu alte cuvinte, definește apeluri către server pentru:

- descrierea conținutului câmpurilor returnate pe baza informațiilor din dicționarul de date, pentru analiza (*parse*) frazelor SQL din punct de vedere sintactic;
- dechiderea unui cursor în urma unei fraze SQL;
- executarea frazelor SQL în spațiul de memorie destinat cursorului creat;
- aducerea (*fetch*) unei înregistrări sau a mai multor înregistrări în aplicația-client;
- închiderea cursorului creat în scopul execuției frazei SQL.

Aplicația-client utilizează o combinație a acestor apeluri pentru a trimite și executa cereri pe serverul ORACLE.

Crearea unei surse de date Oracle prin ODBC

Să clarificăm mai întâi la ce ne referim când vorbim despre o sursă de date sau despre un client ODBC pentru un server BD.

În sens larg, o sursă de date constă din datele (baza de date) pe care un utilizator dorește să le acceseze, plus informațiile necesare pentru accesarea lor, referitoare la sistemul de operare, SGBD și substratul de transport în rețea.

Clientul poate fi asimilat unei aplicații care accesează date de pe un server utilizând o interfață corespunzătoare. Serverul recepționează, procesează cererile clientului și transmite rezultatul clientului care a inițiat cererea. Un client ODBC utilizează interfața API a driverului ODBC pentru a apela funcțiile specifice acesteia în scopul transmiterii frazelor SQL către serverele de date și recepționării/primirii rezultatelor acestor cereri.

Pentru a crea o sursă de date Oracle8i accesibilă prin `Oracle ODBC Driver`, trebuie urmați mai mulți pași.

Mai întâi, trebuie apelat administratorul surselor de date ODBC (din `Control Panel - ODBC Data Source`). Din cadrul de pagină `(User/System/File) DataSourceName` (diferențierea pe *user*, *system* sau *file* este proprie NT-ului și se referă la domeniul de vizibilitate al sursei de date pe mașina respectivă) se alege butonul `Add...`, care lansează fereastra de dialog `Add Data Source`. Din lista de drivere ODBC se alege `Oracle ODBC Driver` (care va fi disponibil o dată cu instalarea clientului Oracle pe respectiva mașină). De regulă, există și drivere pentru Oracle create de Microsoft (care funcționează în mare măsură identic).

Pasul imediat următor presupune completarea informațiilor cerute în fereastra de dialog `Oracle ODBC Setup` (sau `Microsoft ODBC Driver for Oracle Setup` ori `Microsoft ODBC for Oracle Setup` pentru driverele furnizate de Microsoft) – vezi figura 16.4. Dintre aceste informații, esențiale sunt cele legate de: `Data Source Name`, `UserId` și `Service Name` (sau `Connect String` ori `Server` pentru driverele Microsoft). După selectarea butonului `OK` din ultima fereastră de dialog, numele sursei de date se va regăsi printre celelalte surse de date disponibile pe respectiva mașină.

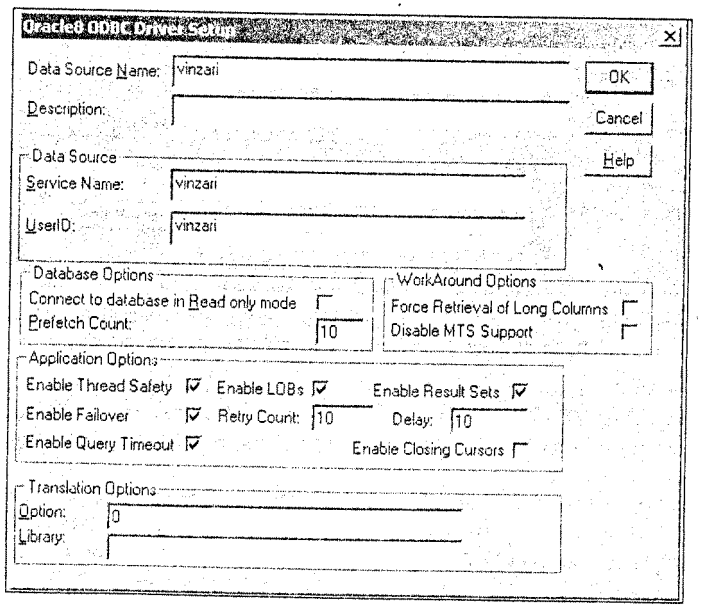


Figura 16.4. Fereastra de configurare a driverului ODBC pentru Oracle

Crearea unei conexiuni VisualFoxPro (client Oracle)

O conexiune (un obiect de tip `connection`) poate fi asimilată definiției unei surse de date stocate într-o bază de date VFP. Această definiție este utilizată atunci când se deschide o sesiune de lucru cu baza de date care este reprezentată prin sursa de date respectivă. Deschiderea unor astfel de sesiuni se realizează fie implicit prin `remote views` (tabele derivate la distanță), fie explicit prin interfața de transfer SPT (SQL Pass-Through).

Definirea unui obiect de tip `connection` într-o BD VFP va cuprinde o serie de parametri care se referă la specificarea sursei de date pentru care este construită conexiunea VFP și o altă serie de parametri care se referă la gestionarea comunicării/traficului dintre clientul VFP și sursa (baza) de date *remote* (la distanță).

Crearea și configurarea unei asemenea conexiuni se poate realiza fie asistat, cu ajutorul `Connection Designer`-ului, ca în figura 16.5, fie „manual", cu ajutorul comenzilor `CREATE CONNECTION` și `DBSETPROP` (parametrii de configurare ai unui obiect de tip `connection` se stochează în dicționarul bazei de date) – vezi listingul 16.5.

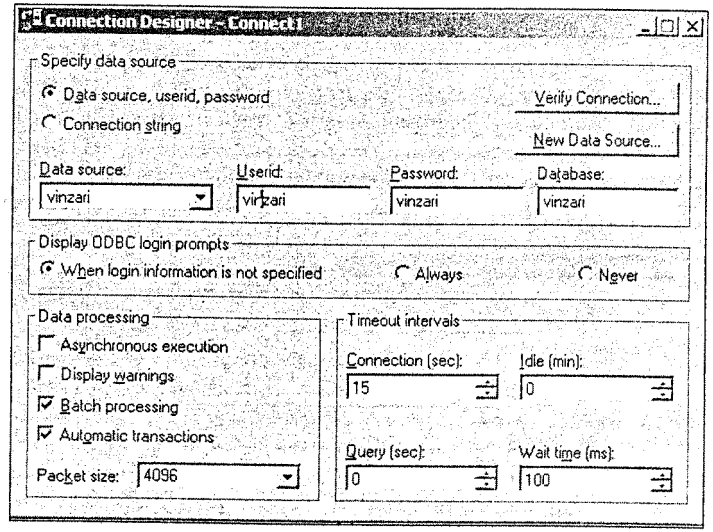


Figura 16.5. Fereastra `Connection Designer` pentru crearea legăturii cu baza de date `VINZARI` localizată pe serverul Oracle

Legarea VFP cu Oracle, chiar în condițiile în care tabelele bazei sunt, în totalitate, gestionate în Oracle, presupune existența sau, în caz contrar, crearea unei BD și pe client, pentru stocarea informațiilor privitoare la conexiuni și a definițiilor eventualelor tabele derivate la distanță.

Precizarea sursei de date se poate face fie prin indicarea explicită a informațiilor referitoare la: `Data source`, `Userid`, `Password`, `Database` (selectând opțiunea `Data source, Userid, Password`), fie prin completarea unui șir de caractere (selectând opțiunea `Connection string`) care se compune, în cazul driverului pentru Oracle, din specificațiile parametrilor: `DSN` pentru numele sursei de date ODBC, `UID` pentru identificatorul utilizatorului în a cărui schemă se găsesc datele ce se doresc a fi

accesate, `PWD` – parola utilizatorului, `DBQ` pentru precizarea bazei de date/serviciului Oracle.

În cazul șirului de caractere pentru conectare, acesta poate fi construit și automat de sistem, prin fereastra de dialog activată la selectarea butonului cu eticheta „..." din dreapta căsuței de tip text aferentă șirului de conectare. Aceste două modalități de specificare a sursei de date sunt echivalente și cu instrucțiunile din listingul 16.5 :

Listing 16.5. Crearea conexiunii prin program

```
CREATE CONNECTION OraVinzari;
DATASOURCE vinzari ;
USERID vinzari PASSWORD vinzari DATABASE vinzari
*** sau
CREATE CONNECTION OraVinzari ;
CONNSTRING 'DSN=vinzari;UID=vinzari;PWD=vinzari;DBQ=vinzari'
```

Specificațiile pentru gestionarea comunicării dintre clientul VFP și baza/sursa de date Oracle se referă în principal la: suport pentru tranzacții, procesare sincronă/asincronă, modul de lucru batch/non-batch.

Tranzacțiile sunt gestionate fie automat, fie manual cu ajutorul interfeței SPT prin funcțiile `SQLCOMMIT()` și `SQLROLLBACK()`.

Modalitățile de prelucrare a datelor din sursele/bazele de date Oracle, sincronă sau asincronă, se referă la modul în care se returnează controlul aplicației după apelarea unei funcții ce implică trafic ODBC.

Modul de lucru `batch` sau `non-batch` stabilește modul de execuție al unei funcții care poate returna mai multe seturi de date dintr-o sursă de date ODBC (ce-i drept, nu am reușit să convingem Oracle să returneze mai multe seturi de date în urma unui singur apel ODBC, cum este posibil cu MS SQLServer).

Aceste specificații au mai mare importanță când se lucrează cu conexiunile ODBC într-un mod mai „intim", prin funcțiile specifice interfeței SPT (*SQL Pass-Through*). Parametrii privitori la aceste specificații se pot configura explicit pentru o conexiune, în dicționarul bazei de date, cu ajutorul funcției `DBSETPROP`. Prin această funcție se pot stabili atributele:

- `transactions`, pentru managementul tranzacțiilor, ce poate lua valorile 1 (`DB_TRANSAUTO` din `foxpro.h`) sau 2 (`DB_TRANSMANUAL` din `foxpro.h`);
- `asynchronous`, pentru care valoarea `.f.` specifică o conexiune sincronă, iar `.t.` una asincronă;
- `batchmode`, care setat pe `.t.` indică o conexiune care lucrează în modul batch.

16.4. Actualizarea tabelelor din baza de date gestionată de către serverul Oracle

Pentru a regăsi/interoga sau modifica datele stocate în tabelele relaționale ale serverului Oracle se poate apela la două mecanisme: primul se bazează pe *tabele derivate la distanță – remote data view*, iar cea de-a doua pe dialogul *SQL Pass-Through*, care oferă flexibilitatea unui acces mai intim asupra canalului și structurilor de date prin care se face legătura cu serverul.

16.4.1. Tabele derivate la distanță

O tabelă derivată, așa cum am descris în capitolul 9, reprezintă un tip de tabelă care se bazează pe o frază SELECT-SQL care se execută în momentul materializării acesteia. Prin urmare, în cazul unui *remote view*, tabelele de bază sunt stocate în altă bază de date decât cea în care este definită, materializarea ei (adică regăsirea înregistrărilor provenind din baza de date sursă) realizându-se prin execuția frazei SELECT-SQL asociate folosind un canal ODBC, accesibil în VFP prin intermediul obiectelor de tip CONNECTION.

Pentru exemplificare, creăm o tabelă derivată denumită VPERSOANE prin care în VFP se vizualizează și (mai ales) actualizează tabela PERSOANE din Oracle. În listingul 16.6 este prezentat programul de configurare a tabelei derivate la distanță, asemănătoare celor pentru tabelele derivate locale (vezi capitolul 9).

Listing 16.6. Crearea tabelei derivate la distanță VMATERIALE

```
**** Creare REMOTE VIEW pentru tabela Vinzari.PERSOANE
**** de pe serverul Oracle
#include foxpro.h
CREATE SQL VIEW vPersoane CONNECTION OraVinzari ;
  AS SELECT * FROM Persoane

=DBSETPROP('vPersoane','View','Tables','Persoane')
=DBSETPROP('vPersoane.CNP','Field','KeyField',.T.)
=DBSETPROP('vPersoane.CNP','Field','Updatable',.T.)

=DBSETPROP('vPersoane.nume','Field','Updatable',.T.)
=DBSETPROP('vPersoane.prenume','Field','Updatable',.T.)
=DBSETPROP('vPersoane.adresa','Field','Updatable',.T.)
=DBSETPROP('vPersoane.codpost','Field','Updatable',.T.)
=DBSETPROP('vPersoane.email','Field','Updatable',.T.)
=DBSETPROP('vPersoane.sex','Field','Updatable',.T.)
=DBSETPROP('vPersoane.telacasa','Field','Updatable',.T.)
=DBSETPROP('vPersoane.telbirou','Field','Updatable',.T.)
=DBSETPROP('vPersoane.telmobil','Field','Updatable',.T.)

=DBSETPROP('vPersoane','View','UpdateType',DB_UPDATE)
=DBSETPROP('vPersoane','View','WhereType', DB_KEYANDMODIFIED)
=DBSETPROP('vPersoane','View','SendUpdates',.T.)
```

Parametrului WhereType i-a fost atribuită valoarea DB_KEYANDMODIFIED. Aceasta înseamnă că, la „comiterea" bufferului în tabela-sursă, se verifică dacă, între timp, au fost modificate de la altă stație de lucru atributul (atributele) din cheia primară și atributele modificate local. Dacă da, se declanșează eroarea Update conflict.

Observație: Mai toate valorile din WhereType funcționează rezonabil (în general!), nu însă și DB_KEYANDUPDATABLE. Această valoare face ca în mai toate momentele de actualizare a tabelei la distanță să se declanșeze Update conflict. Nu am găsit o explicație logică pentru acest comportament, mai ales că pentru o parte din liniile modificate local, eroarea nu se declanșează !

Toate proprietățile tabelei derivate sunt memorate în dicționarul (containerul) BD. Deschiderea și deci materializarea ei se realizează obișnuit prin comanda:

```
USE vMateriale IN 0
```


O atenție deosebită merită însă modul de propagare a modificărilor operate în tabela derivată către tabela-mamă din Oracle. În mod obișnuit, o tabelă derivată la distanță prezintă un mod de lucru de tip Optimistic Row Buffering. Aceasta înseamnă că modificarea unei linii se „comite" în tabela de bază atunci când pointerul VFP al tabelei derivate se poziționează pe o altă înregistrare sau la execuția funcției TABLEUPDATE().

Pe de altă parte, actualizarea tabelei derivate pe baza modificărilor de ultim moment apărute în BD Oracle se realizează prin funcția REQUERY().

În dezvoltarea aplicațiilor este necesară preluarea și tratarea erorilor provocate de către tabela derivată bazei de date Oracle. Figura 16.6 ilustrează un caz de violare (prin VPERSOANE) a cheii primare aparținând tabelei PERSOANE.

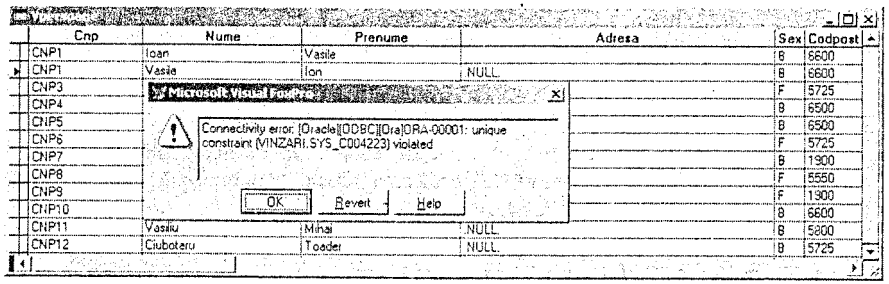


Figura 16.6. Semnalizarea violării cheii primare

Valoarea care se repetă pentru câmpul CNP este CNP1. Din mesajul afișat se poate observa codul Oracle al erorii (-00001). Problema este: cum preluăm într-o aplicație VFP, în partea sa de client, acest mesaj și cum rezolvăm problema fără combinația salvatoare CTRL+ALT+DEL?

Soluția vine de la funcția AERROR(). Aceasta plasează într-un vector cu numele indicat informații despre ultima eroare produsă. Pentru erorile ODBC, fiecare component al masivului este descris în tabelul 16.1

Tabelul 16.1. Descrierea informațiilor obținute prin funcția AERROR()

Component	Tip și conținut
1	Numeric. Conține valoarea 1526 (Connectivity error:...), indiferent de cauza erorii ODBC
2	Caracter. Textul mesajului de eroare
3	Caracter. Textul mesajului de eroare ODBC
4	Caracter. Starea curentă SQL ODBC
5	Numeric. Numărul erorii preluat de la sursa ODBC
6	Numeric. Numărul conexiunii ODBC
7	Valoarea NULL

Dacă, de exemplu, la editarea tabelei derivate VPERSOANE, numele unei persoane va începe cu literă mică, în momentul comiterii modificării se generează o eroare care, preluată prin funcția =AERROR(vErr), inițializează masivul vErr, după cum este prezenat în tabelul 16.2:

Tabelul 16.2. Descrierea conținutului masivului vErr

Componenta vErr	Conținut
vErr(1)	1526
vErr(2)	Connectivity error: [Oracle][ODBC][Ora]ORA-02290: check constraint (VINZARI.SYS_C004219) violated
vErr(3)	[Oracle][ODBC][Ora]ORA-02290: check constraint (VINZARI.SYS_C004219) violated
vErr(4)	23000
vErr(5)	2290
vErr(6)	1
vErr(7)	NULL

În baza de date Oracle a fost definită, pentru atributul Nume din tabela PERSOANE, o restricție (prin clauza CHECK nume=LTRIM(INITCAP(nume)) prin care valorile acestui atribut trebuie să fie șiruri de caractere care încep cu majusculă, fără spații la început. Eroarea Oracle declanșată la încălcarea acestei restricții are numărul 2290 (check constraint violated).

16.4.2. Tehnogia *SQL Pass-Trough* – SPT

Cea de-a doua modalitate de a lega aplicații VFP la servere de baze de date o constituie tehnologia *SQL Pass-Trough* (SPT).

Dezavantajul, prin comparație cu tabelele derivate la distanță, ține de faptul că efortul de codare este mult mai mare. Fiecare operațiune prezintă funcții și comenzi destul de laborioase. În plus, după cum am văzut, tabela derivată poate fi legată direct la controalele unui formular, propagarea modificărilor făcându-se automat, în timp ce în SPT orice actualizare trebuie operată explicit.

Avantajul net ține de flexibilitate. Dacă în cazul tabelelor derivate doar triggerele Oracle puteau fi declanșate prin modificarea datelor (și, automat, propagarea modificărilor în tabelele serverului BD), utilizând SPT, nu numai că gestiunea conexiunii VFP-Oracle este una mult mai analitică, dar se poate invoca explicit orice procedură stocată Oracle și chiar pot fi utilizate comenzi DDL. De asemenea, există posibilitatea controlului „manual" al tranzacțiilor serverului.

Interogările SPT creează implicit un cursor (o copie neactualizabilă a datelor extrase din BD Oracle). Cursorul poate fi făcut actualizabil prin setarea parametrilor cu ajutorul funcției CURSORSETPROP().

Stabirea conexiunii

Ca și în cazul tabelelor derivate la distanță, mai întâi este necesară stabilirea unei conexiuni ODBC. Legarea de BD Oracle se realizează prin funcția SQLCONNECT() sau SQLSTRINGCONNECT(). Prin SQLCONNECT() fără argumente se afișează o fereastră de dialog prin care utilizatorul își poate alege una dintre sursele de date. SQLDISCONNECT() dezactivează conexiunea cu BD aflată pe server. În exemplul următor, variabila nrConexiune va conține numărul conexiunii stabilite (mai mare decât

zero). Atunci când legătura nu poate fi stabilită, valoarea nrConexiune este mai mică decât 1.

```
nrConexiune = SQLCONNECT('OraVinzari')
IF nrConexiune < 1
    =MESSAGEBOX("Conexiune esuata",0," Rezultat tentativa")
    RETURN
ENDIF
```

Interogarea BD și apelul procedurilor stocate Oracle

Pentru a accesa date aflate pe serverul BD, este necesară declararea unui canal/magistrală (traducere aproximativă a lui handle) pentru conexiunea activă. O dată stabilită legătura, prin funcția SQLEXEC(), din VFP pot fi lansate comenzi Oracle DML și DDL și pot fi apelate proceduri stocate. În miniexemplul următor, din VFP, prin fraza SELECT se interoghează tabela Oracle PERSOANE.

```
vSucces = SQLEXEC(nrConexiune, 'select * from Persoane', 'cPersoane')
IF vSucces > 0
SELECT cPersoane
BROWSE
ELSE
    =MESSAGEBOX("Interogare fara rezultat ",0," Rezultatul interogarii ")
ENDIF
```

Variabila vSucces va conține numărul de seturi de rezultate furnizat de serverul BD (în mod obișnuit 1) sau 0, dacă funcția este încă în curs de execuție. Setul de rezultate furnizat de Oracle va deveni în VFP un cursor denumit CPERSOANE.

În dezvoltarea aplicațiilor, pentru optimizarea traficului pe rețea, deosebit de important este ca interogările să fie parametrizate. Exemplu:

```
CodPost_ = 6600
vSucces = SQLEXEC(nrConexiune, 'SELECT * FROM Persoane WHERE codpost = ?codpost_'
,; 'cPersoane')
SELE cPersoane
BROWSE
```

Variabila-parametru codpost_ este folosită pentru a extrage din tabela Oracle PERSOANE numai înregistrările persoanelor din localitatea având codul poștal 6600. Firește, această variabilă poate fi utilizată în formular pentru ca în grid-uri, combo-box-uri etc. să fie afișate un minim de date, deci fără aglomerarea rețelei.

Invocarea unei proceduri stocate de pe serverul BD este una dintre cele mai importante facilități ale tehnologiei SPT prin comparație cu tabelele derivate la distanță. Să presupunem că dorim să efectuăm operațiuni de ștergere înregistrări din tabela PERSOANE aflată în BD Oracle nu prin intermediul unei tabele derivate la distanță sau trimițând comanda DELETE-SQL printr-un dialog SPT, ci invocând o procedură stocată care va fi executată pe serverul Oracle. În listingul 16.7 este prezentată comanda pentru crearea acestei proceduri stocate, comandă ce poate fi executată din utilitarul SQL*Plus, bineînțeles, după ce în prealabil ne-am conectat la schema Vinzari.

Listing 16.7. Procedura stocată din BD Oracle pentru ştergere înregistrări din tabela PERSOANE

```
CREATE OR REPLACE
Procedure STERG_PERSOANA
  ( pCNP IN CHAR, pRezultat OUT VARCHAR2)
  IS
violare_fk exception;
v_client number;
pragma exception_init(violare_fk, -02292);
BEGIN
  DELETE FROM Persoane WHERE CNP = pCNP;
  pRezultat := 'Succes';
EXCEPTION
  WHEN violare_fk THEN
  SELECT codcl INTO v_client FROM PersClienti WHERE CNP = pCNP;
  DELETE FROM PersClienti WHERE CNP = pCNP;
  DELETE FROM Persoane WHERE CNP = pCNP;
  pRezultat := 'A fost stearsa persoana de contact pentru clientul '||v_client;
  WHEN OTHERS THEN
  pRezultat := 'Insucces';
END; -- Procedure STERG_PERSOANA
```

Prin procedura `Sterg_Persoana` se va încerca îndepărtarea din BD a înregistrării al cărei CNP este primit ca parametru. În cazul în care se constată violarea restricţiei referenţiale dintre tabelele `PERSOANE` şi `PERSCLIENTI`, se şterge mai întâi înregistrarea-copil corespunzătoare.

Invocarea unei proceduri stocate se face în mod obişnuit în felul următor:

```
=SQLEXEC(nrConexiune, "BEGIN nume_procedura_stocată; END;")
```

Cu alte cuvinte, transmitem serverului Oracle un bloc PL/SQL anonim în care este apelată procedura stocată dorită.

O problemă ridicată de apelul procedurilor şi funcţiilor stocate Oracle o constituie transmiterea parametrilor şi preluarea rezultatelor returnate de funcţii. Procedura creată prin listingul anterior are un parametru de intrare `pCNP`, prin care se transmite CNP-ul persoanei care urmează să fie ştearsă, şi un parametru `pRezultat` din care putem deduce ce a reuşit să realizeze serverul Oracle ca urmare a apelului nostru. În listingul 16.8 este exemplificat apelul procedurii stocate `Sterg_Clienti`, ţinând seama de parametrii acesteia.

Listing 16.8. Invocarea unei proceduri stocate Oracle cu parametri de intrare

```
nrConexiune = SQLCONNECT('OraVinzari')
IF nrConexiune < 1
    =MESSAGEBOX("Conexiune esuata",0," Rezultat tentativa")
    RETURN
ENDIF

*** preluarea valorilor variabilelor-parametri ai procedurii Oracle
vCNP = 'CNP1'
vRezultat = 'Incert'
nRezultat = SQLEXEC(nrConexiune,"BEGIN sterg_persoana(?vCNP, ?@vRezultat) ; END ;")
IF nRezultat < 0
    ** operatiune esuata
    messagebox('Problema')
    DIME vEroare(1,1)
    =AERROR(vEroare)
```



```
    * secventa de tratare a erorii
    *
ELSE
    Messagebox(vRezultat)
ENDIF

=SQLDISCONNECT(nrConexiune)
nrConexiune = SQLCONNECT('OraVinzari')
IF nrConexiune < 1
    =MESSAGEBOX("Conexiunea n-a mai putut fi restabilita !",0," Rezultat tentativa")
    RETURN
ENDIF
=SQLEXEC(nrConexiune,'SELECT * FROM Persoane','cPersoane')
SELECT cPersoane
browse
```

Variabila `vCNP` a fost iniţializată în programul VFP având ca scop transmiterea valorii CNP-ului vizat procedurii stocate de pe serverul Oracle prin intermediul apelului `SQLEXEC()`. De asemenea, rolul variabilei `vRezultat` a fost de a prelua valoarea parametrului de ieşire al procedurii `Sterg_Clienti`. Fireşte că, în aplicaţii, astfel de variabile pot fi legate la controale ale formularelor VFP. Este posibil ca execuţia procedurii să se soldeze cu erori din partea BD Oracle. Un prim indiciu în acest sens îl reprezintă variabila `nRezultat`. Dacă în urma execuţiei `SQLEXEC`, `nRezultat` este `-1`, înseamnă că există o eroare. Preluarea erorii în vederea tratării se realizează prin funcţia `AERROR()`.

`SQLDISCONECT()` şi secvenţa care o urmează au drept scop reîmprospătarea cursorului. Reţineţi sintaxa folosită pentru a transmite valoarea unei variabile locale în apelul unei proceduri stocate (`?nume_variabilă`) şi pentru preluarea valorii returnate de o procedură stocată printr-un parametru de ieşire (`?@nume_variabilă`).

Un interes aparte îl prezintă şi preluarea în variabile Fox a rezultatului unei funcţii PL/SQL. Pentru ilustrare, creăm o funcţie denumită `Tel_PersContact_Client`. Această funcţie are trei parametri de intrare prin care se precizează codul clientului, funcţia persoanei de contact, iar în la treilea se va prelua numele acesteia. Scopul este de a returna informaţii prin care poate fi contactată persoana de legătură cu un anumit client (vezi listingul 16.9).

Listing 16.9. Comanda pentru crearea funcţiei `Tel_PersContact_Client`

```
CREATE OR REPLACE
Function TEL_PERSCONTACT_CLIENT
  ( pCodCl  IN NUMBER, pFunctie IN VARCHAR2, pNume OUT VARCHAR2)
 RETURN VARCHAR2
IS
vTelefon VARCHAR2(100);
BEGIN
   SELECT 'Acasa:'||DECODE(telacasa, NULL, 'Necunoscut', telacasa)||
   ';Birou:'||DECODE(telbirou, NULL, 'Necunoscut', telbirou)||
   ';Mobil:'||DECODE(telmobil, NULL, 'Necunoscut', telmobil),
   Nume||' '||Prenume
   INTO vtelefon, pNume
   FROM persoane p, persclienti c
   WHERE p.cnp = c.cnp and c.codcl = pCodCl and c.functie = pFunctie;
   RETURN vTelefon;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
   RETURN 'No contact';
END; -- Function CONTACT_CLIENT
/
```

Apelul şi preluarea rezultatului returnat de această funcţie se realizează după cum este prezentat în listingul 16.10.

Listing 16.10

```
nrConexiune = SQLCONNECT('OraVinzari')
IF nrConexiune < 1
    =MESSAGEBOX("Conexiune esuata",0," Rezultat tentativa")
    return
ENDIF
v_telcontact= ''              && initializam "manual" variabilele
v_codcl = 1002
v_functie = 'Director general'
v_nume = ''
sir_="DECLARE "+;
        "R VARCHAR2(100); "+;
        "N VARCHAR2(41); "+;
    "BEGIN "+;
    " ?@v_telcontact := TEL_PERSCONTACT_CLIENT(?v_codcl, ?v_functie, ?@v_nume);"+
    "END;"
nRezultat=SQLEXEC(NrConexiune,sir_)
IF nRezultat < 0
    ** operatiune esuata
    messagebox('eRR')
    dime vEroare(1,1)
    =aerror(vEroare)
    *
    * secventa de tratare a erorii
    *
ELSE
    * operatiune reusita
    messagebox(v_nume + CHR(13) + v_telcontact)
ENDIF
```

Funcţia VFP `SQLEXEC()`, ce realizează invocarea funcţiei Oracle, conţine un minibloc PL/SQL. Pentru claritate, am compus şirul de caractere care va fi transmis serverului Oracle prin intermediul variabilei `sir_`. Esenţială în preluarea rezultatului din Oracle este variabila `v_telcontact`, ce trebuie musai prefixată de ambele caractere, `?@`.

16.4.3. Cursoare actualizabile

În dezvoltarea aplicaţiilor VFP-Oracle, dincolo de invocarea procedurilor şi funcţiilor stocate pe server, important este şi modul de propagare a modificărilor operate în tabelele/cursoarele VFP în baza de date aflată pe server. Un cursor obţinut printr-un `SELECT` trimis serverului Oracle prin `SQLEXEC()` poate fi declarat ca modificabil într-o manieră nu cu mult deosebită de tabelele derivate. Setarea parametrilor de actualizare se face nu prin `DBSETPROP()`, ci prin `CURSORSETPROP()`.

Spre deosebire de tabelele derivate la distanţă, definiţia cursoarelor actualizabile nu este stocată în dicţionarul (containerul) bazei de date, astfel încât programul de creare/setare trebuie executat la fiecare deschidere a cursorului.

În listingul 16.11, cursorul VFP `CPERSOANE` obţinut din tabela Oracle `PERSOANE` este declarat actualizabil.

Listing 16.11. Crearea unui cursor VFP actualizabil

```
**** Creare cursor actualizabil pentru tabela Vinzari.PERSOANE
**** de pe serverul Oracle
#include foxpro.h

nrConexiune = SQLCONNECT('OraVinzari')
IF nrConexiune < 1
    =MESSAGEBOX("Conexiune esuata",0,"Abandon")
    RETURN
ENDIF
IF used('cpersoane')
 sele cpersoane
 use
endif
vSucces = SQLEXEC(nrConexiune, 'SELECT * FROM Persoane', 'cPersoane')
if vSucces < 1
    =MESSAGEBOX("Interogare fara rezultat ",0," Rezultatul interogarii ")
    RETURN
endif

SET MULTILOCKS ON
* pentru cursor se declara bufferingul
* de tip ROW OPTIMISTIC
=CURSORSETPROP("Buffering",3,"cPersoane")
=CURSORSETPROP('Tables','Persoane','cPersoane')
=CURSORSETPROP('KeyFieldList','CNP','cPersoane')
=CURSORSETPROP('UpdatableFieldList','CNP,Nume,Prenume,Adresa,CodPost,;
  email,sex,telacasa,telbirou,telmobil','cPersoane')
* maparea atributelor cursorului la atributele tabelei de baza
=CURSORSETPROP('UpdateNameList','CNP Persoane.CNP,Nume Persoane.Nume,;
Prenume Persoane.Prenume, Adresa Persoane.Adresa, CodPost Persoane.CodPost,;
email Persoane.email, sex Persoane.sex, telacasa Persoane.telacasa, telbirou Persoane.telbirou,;
telmobil Persoane.telmobil','cPersoane')
=CURSORSETPROP('WhereType',DB_KEY, 'cPersoane')
=CURSORSETPROP('SendUpdates',.T.,'cPersoane')
=CURSORSETPROP('UpdateType',DB_UPDATE,'cPersoane')
```

Din momentul declarării şi editării cursorului, modificările operate vor fi propagate automat în baza de date de pe server. Rămâne de rezolvat problema reîmprospătării lui, pentru a vizualiza şi modificările operate de alţi utilizatori. Cum, pentru cursoare, funcţia `REQUERY()` nu funcţionează, cel mai simplu mod de *refresh* este recrearea, de fiecare dată când se doreşte reîmprospătarea, a `CPERSOANE`, prin execuţia procedurii din listingul 16.7 (grijă la închiderea conexiunii şi redeschiderea canalului).

16.5. Formulare VFP – accesul şi modificarea datelor aflate pe serverul Oracle

În ultima etapă de conversie/migrare a aplicaţiei din VFP în Oracle problemele cele mai deosebite le ridică formularele.

Pentru a „convinge" un formular VFP să lucreze corect cu datele stocate în tabelele Oracle, acesta trebuie supus unui proces de transformare în mai mulţi paşi: în primul rând, definirea sau redefinirea mecanismelor de acces la înregistrările din tabela (tabelele) de

bază acum stocate pe serverul Oracle, redefinirea sursei datelor pentru obiectele de acces la date ale formularului (căsuțele de text, listele combinate, grupurile de butoane-radio etc.), redefinirea mecanismelor de navigare pe înregistrările accesebile prin formular, redefinirea mecanismelor de control al operațiilor de actualizare (inserări, modificări, ștergeri), inclusiv *definirea unui mecanism de tratare a erorilor pentru operațiile neîncheiate cu succes*.

Pentru exemplificare, am recurs la formularul `FrmPersoane` descris pe larg în capitolul 8, subcapitolul 8.2. Acesta va fi transformat după cum urmează:

1) *Sursa de date a formularului*:

- Dacă se lucrează cu `DataEnvironment`, atunci tabela `PERSOANE` (din baza de date locală VFP) va fi scoasă, iar în locul ei va fi introdusă tabela derivată `VPERSOANE`.
- Dacă tabelele de bază sunt gestionate în metoda `Load` a formularului, atunci în codul-sursă al acesteia va fi specificată instrucțiunea `USE` pentru deschiderea tabelei derivate la distanță `VPERSOANE` (vezi listingul 16.12).

Listing 16.12. Metoda `Load` a formularului `FrmPersoane`

```
*-----------------------------------------------------------------------
** Metoda FrmPersoane.LOAD
*-----------------------------------------------------------------------
LOCAL nrConexiune, exista_tag
IF NOT DBUSED('vinzari')
    OPEN DATABASE "\aplicatie\database\vinzari" SHARED
ENDIF
*** Deschid tabela derivata VPersoane care
*** face legatura cu tabela Persoane de pe serverul ORACLE
IF !USED('vpersoane')
 use vpersoane in 0
ENDIF

***** Creez cheia de navigare necesară pentru indexare
select vpersoane
for i=1 to tagcount()
 if ALLT(UPPER(tag(i))) = 'CNP'
  exista_tag = .t.
 endif
endfor
if !exista_tag
 index on cnp tag cnp
endif

*** Stabilesc sursa de date pentru cboCodPost
select vpersoane
nrConexiune = cursorgetprop('connecthandle')
vSucces = SQLEXEC(nrConexiune, "select loc,judet,codpost from localitati l,judete j;
 where l.jud=j.jud  order by loc", "crslocalitati")

IF vsucces<0
  messagebox('Eroare! Nu s-a reusit popularea listei de localitati !!')
  return .f.
endif

*** Stabilesc sursa datelor pentru cboPersoana
vSucces = SQLEXEC(nrConexiune, "select rtrim(ltrim(nume))||' '||rtrim(ltrim(prenume)),;
 cnp from persoane order by nume,prenume", "crspersoane")
```



```
IF vsucces<0
  messagebox('Eroare! Nu s-a reusit popularea listei de persoane !!')
  return .f.
endif
```

2) *Sursele de date pentru obiectele de acces*:

- Pentru căsuțele de text `txtCNP`, `txtNume`, `txtPrenume`, `txtAdresa`, `txtTelacasa`, `txtTelmobil`, `txtTelbirou`, `txtEmail` proprietățile `Control Source` vor avea valorile: `VPersoane.CNP`, `VPersoane.Nume`, `VPersoane.Prenume`, `VPersoane.Adresa`, `VPersoane.Telacasa`, `VPersoane.Telmobil`, `VPersoane.Email`.
- Pentru lista combinată `cboPersoana`, proprietatea `RowSourceType` va avea valoarea `Alias`, iar `RowSource` va avea valoarea `crsPersoane`. La fel și pentru lista combinată `cboCodPost`: `RowSourceType = Alias`, iar `RowSource = crsLocalități`. Cele două cursoare se formează în metoda `Load` a formularului ca urmare a execuției comenzilor `SQLEXEC` ce invocă frazele `SELECT-SQL` corespunzătoare (vezi listingul 16.12). Pentru execuția acestor interogări am folosit același canal-conexiune ca și cel utilizat pentru deschiderea tabelei derivate `VPERSOANE`, canal al cărui număr l-am aflat citind proprietatea `ConnectHandle` a cursorului acesteia.
- Gestiunea grupului de butoane-radio `OptGrupSex`, care redă și modifică informația din câmpul `Sex`, implică modificarea metodelor `Refresh` și `InteractiveChange` (vezi listingul 16.13).

Listing 16.13. Metodele controlului `OptGrupSex`

```
*-----------------------------------------------------------------------
** Metoda OptGrupSex.Refresh
*-----------------------------------------------------------------------
IF vpersoane.sex='B'
    THIS.btnB.VALUE=1
    THIS.btnF.VALUE=0
ELSE
    THIS.btnB.VALUE=0
    THIS.btnF.VALUE=1
ENDIF

*-----------------------------------------------------------------------
** Metoda OptGrupSex.InteractiveChange
*-----------------------------------------------------------------------
IF THIS.btnB.VALUE=1
    REPLACE vPERSOANE.SEX WITH 'B'
ELSE
    REPLACE vPERSOANE.SEX WITH 'F'
ENDIF
*-----------------------------------------------------------------------
```

- Navigarea în formularul `FrmPersoane` se realizează prin intermediul listei `cboPersoane` și butoanelor de navigare din grupul `CmdGrupNavigare` (care însă se folosesc tot de comportamentul asociat listei `cboPersoane`). Acest control invocă în metoda `InteractivChange` comanda `SEEK` pentru sincronizarea tabelei de bază. Comanda `SEEK` se bazează pe o cheie ce aparținea inițial tabelei

PERSOANE din baza de date VFP. Cum indecşii din Oracle nu pot fi invocaţi în astfel de operaţiuni, soluţia pe care am putea-o folosi ar fi facilitatea VFP de a defini chei de indexare şi pentru tabele derivate. Aşa se explică prezenţa comenzii INDEX ON din metoda Load a formularului (vezi listingul 16.12).

3) Mecanismul tranzacţional (BEGIN TRANSACTION ... END TRANSACTION) şi operaţiile de inserare/modificare/ştergere, controlate prin butoanele corespunzătoare, rămân în linii mari aceleaşi, dar cu următorul amendament: în metodele click – pentru butonul cmdOk şi Destroy – de la nivelul formularului (vezi listingul 16.14) vom invoca, în scopul trimiterii modificărilor de pe clientul VFP pe serverul Oracle, o metodă nouă, definită de noi, şi anume Scrie_Buffer (vezi listingul 16.15).

Listing 16.14. Metodele cmdOk.click şi FrmPersoane.Destroy

```
*-----------------------------------------------------------------------------
** Metoda cmdOK.Click
*-----------------------------------------------------------------------------
END TRANSACTION
select vpersoane
thisform.scrie_buffer
thisform.refresh

THISFORM.dezactivare
THISFORM.CBOPERSOANA.REQUERY
FOR I=1 TO THISFORM.CBOPERSOANA.LISTCOUNT
    IF THISFORM.CBOPERSOANA.LIST(I,2)==vpersoane.cnp
        THISFORM.CBOPERSOANA.LISTINDEX=I
    ENDIF
ENDFOR

*-----------------------------------------------------------------------------
** Metoda FormPersoane.Destroy
*-----------------------------------------------------------------------------

IF TXNLEVEL()>0
    IF MESSAGEBOX('Salvati ultimele modificari?',32+4,'Ultimele modificari nu au fost validate')=6
        END TRANSACTION
    ELSE
        ROLLBACK
    ENDIF
ENDIF
IF THISFORM.asters=.T.
    SELECT persoane
    thisform.scrie_buffer
ENDIF
*-----------------------------------------------------------------------------
```

Metoda Scrie_Buffer încearcă să scrie conţinutul bufferului VFP (înregistrarea modificată a tabelei derivate) în tabela PERSOANE din Oracle. Dacă operaţiunea reuşeşte, variabila a are valoarea TRUE(.T.). În caz contrar, prin funcţia =AERROR(vErr) se iniţializează masivul vErr cu informaţii despre eroare. Întrucât valorile returnate sunt diferite, în funcţie de cauza erorii, este necesară conversia din valori nule şi numerice în şiruri de caractere pentru afişare şi stocarea în tabela TEMP – o tabelă utilizată pentru jurnalizarea erorilor de acest tip, creată prin comanda
CREATE TABLE temp (nr INTEGER, text CHAR (250)).

Listing 16.15. Metoda Scrie_Buffer

```
select vPersoane
rele all like sir*
a = tableupdate()
if !a
    =aerror(vErr)
    for i = 3 to 7                  && conversia in sir de caractere a
                                    && ultimelor cinci componentele ale vectorului
        ii=str(i,1)
        sir&ii = NVL(vErr(i),"NULL")
        sir_ = sir&ii
        if TYPE("sir_") = "N"
            sir&ii = str(sir&ii,10)
        endif
    endfor
        insert into temp values (1, str(vErr(1),7))
        insert into temp values (2, vErr(2))
        insert into temp values (3, sir3)
        insert into temp values (4, sir4)
        insert into temp values (5, sir5)
        insert into temp values (6, sir6)
        insert into temp values (7, sir7)
    do case
        case vErr(1) = 1585                 && Update conflict
            messagebox('Inregistrarea actualizate a fost modificata intre timp de altcineva !')
        case vErr(1) = 1526 and vErr(5) = 1 and "PK_PERSOANE" $ vErr(2)
            messagebox('Se repeta valoarea atributului CNP!')
        case vErr(1) = 1526 and vErr(5) = 1438
            messagebox('Valoarea unui atribut numeric depaseste limitele !')
        case vErr(1) = 1526 and vErr(5) = 2290 and "CK_CNP" $ vErr(2)
            messagebox('Codul numeric personal se scrie fara spatii la inceput !')
        case vErr(1) = 1526 and vErr(5) = 2290 and "CK_ADRESA" $ vErr(2)
            messagebox('Prima litera din adresa este obligatoriu majuscula !')
        case vErr(1) = 1526 and vErr(5) = 2290 and "CK_NUME" $ vErr(2)
            messagebox('Prima litera din fiecare cuvint al'+CHR(13)+;
            'numelui este majuscula; '+CHR(13)+;
            'restul literelor sunt mici!')
        case vErr(1) = 1526 and vErr(5) = 2290 and "CK_PRENUME" $ vErr(2)
            messagebox('Prima litera din fiecare cuvint al'+CHR(13)+;
            'prenumelui este majuscula; '+CHR(13)+;
            'restul literelor sunt mici!')
        case vErr(1) = 1526 and vErr(5) = 2290 and "CK_SEX" $ vErr(2)
            messagebox('Atributul Sex poate avea valorile F (de la Femeiesc)'+CHR(13)+;
                    'sau B (de la Barbatesc) !')
        case vErr(1) = 1526 and vErr(5) = 20015     && extragem mesajul de eroare din
                                                    && exceptia definita in Oracle
            messagebox(substr(vErr(3),31,65))
        otherwise
            messagebox('Eroare netratata !')
    endcase
    =tablerevert()
endif
```

Încălcarea unei restricţii din baza de date va fi prezentată utilizatorului sub forma unui mesaj inteligibil (vezi figura 16.7):

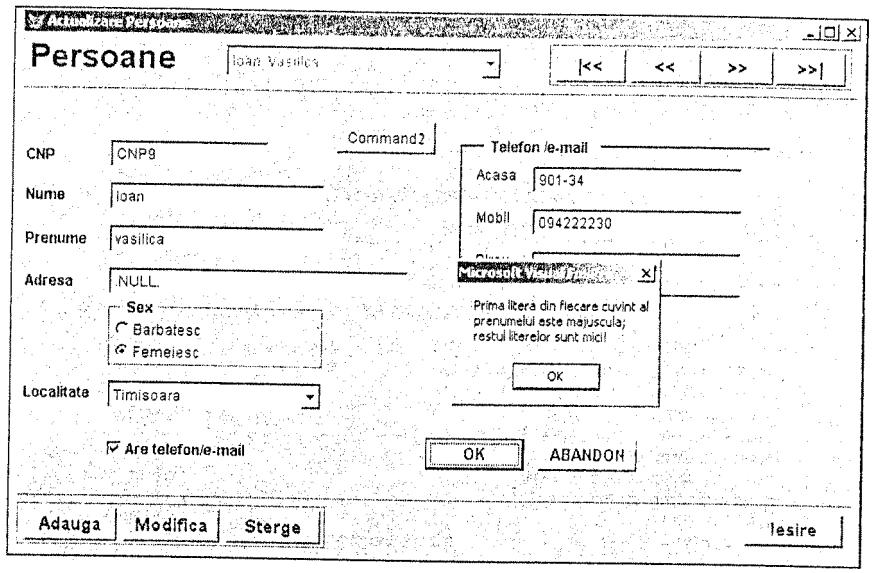


Figura 16.7. Formularul `FrmPersoane` accesând tabela `PERSOANE` de pe serverul Oracle şi tratând o eroare returnată de acesta

Conversia rapoartelor astfel încât să folosească sursele de date stocate pe serverul Oracle este relativ simplă dacă acestea au fost construite respectând recomandarea făcută în capitolul 7, şi anume structura de date din spatele raportului să se bazeze pe una sau mai multe fraze `SELECT-SQL`. Ceea ce ar mai rămâne de făcut în acest caz este traducerea frazelor `SELECT-SQL` specifice VFP în sintaxa Oracle (sau o sintaxă comună ODBC) şi executarea acestora folosind instrucţiunea `SQLEXEC` într-un dialog SPT.

Migrarea la un sistem cu o arhitectură *client VFP – server de baze de date* (eventual Oracle) este o operaţie destul de delicată, însă capabilităţile driverelor ODBC simplifică destul de mult acest proces în special în ceea ce priveşte canalul de legătură. Mai rămân însă de rezolvat sarcini specifice legate de conversia bazei de date şi conversia structurilor de interfaţă. Existenţa unor instrucţiuni `DDL-SQL` drept sprijin pentru construirea bazei de date VFP şi a surselor de date pentru formulare şi rapoarte poate face acest proces mai puţin „dureros". Oricât de complicate ar fi problemele care ar putea apărea la un moment dat, în funcţie de rata de creştere a bazei de date VFP şi a bugetului IT al organizaţiei, trecerea la un astfel de model este inevitabilă. Am mai putea aminti aici şi că un sistem e-business robust are în spate o bază de date din clasa Oracle, DB2 sau MS SQL-Server.

Capitolul 17

Livrarea produsului finit

Lansarea în exploatare constituie, fără îndoială, unul dintre momentele „spumoase" ale dezvoltării oricărui sistem informatic. Este faza în care, după ce a trecut cu succes testele echipei de dezvoltare, produsul este „împachetat", documentat şi livrat utilizatorilor finali.

17.1. Crearea unui sistem de asistenţă. HTML Help Workshop

O aplicaţie complexă (Visual FoxPro sau de altă natură) nu se poate considera completă dacă nu dispune de o documentaţie minimă referitoare la modul de folosire. În epoca microcalculatoarelor cu memorie de ordinul sutelor de Mbytes şi cu hard-disk-uri de zeci de Gbytes (care probabil vor părea modeste peste cinci ani), a Internetului şi altor tehnologii de vârf, recursul la manualul în format clasic este completat, de cele mai multe ori chiar suplinit, de documentaţia în format electronic.

În versiunea anterioară a pachetului Developer Studio (1997), Microsoft a inclus un instrument pentru realizarea fişierelor de ajutor într-un format compatibil cu sistemul Help al Windows 95 (figura 17.1 – cu afişarea fişierelor de asistenţă se ocupă aplicaţia *Winhlp32.exe*). Acest instrument, numit *Help Workshop* (figura 17.2), încă se livrează cu pachetul Visual Studio. Este considerat uşor învechit şi de aceea nu vom oferi informaţii suplimentare asupra modului de utilizare.

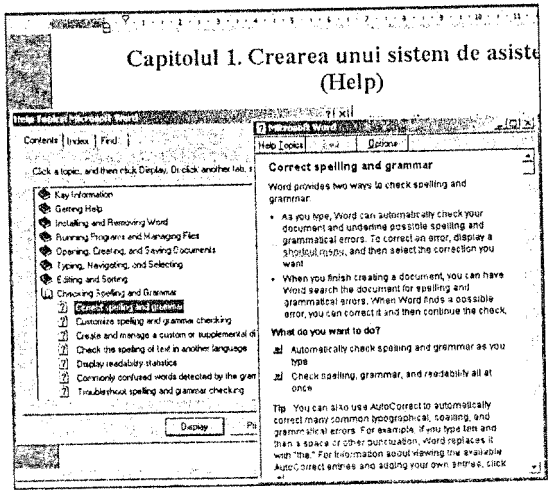


Figura 17.1. Asistenţă on-line (Word 97) în formatul suportat de aplicaţia `Winhlp32`

O dată cu lansarea pachetului Visual Studio 6 (1998), a sistemelor de operare Windows 98 şi Windows 2000, Microsoft a introdus un nou format al documentaţiei, mai apropiat de aspectul şi capacităţile paginilor Web (vezi figura 17.3). Dacă aţi lucrat până acum cu Visual FoxPro, Visual Basic, Visual C++ ori altă aplicaţie Windows recentă, v-aţi familiarizat cu navigarea în acest sistem de *help*. De aceea, nu vom oferi detalii asupra folosirii unei astfel de documentaţii, ci asupra modului cum puteţi dumneavoastră înşivă să scrieţi, adnotaţi şi compilaţi una asemănătoare. Vom spune doar că, „responsabil" de afişarea *help*-ului în acest format este programul *Hh.exe*, care împrumută câte ceva din funcţionalitatea browserului Internet Explorer.

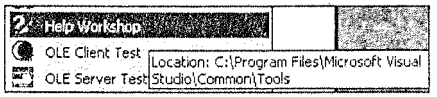


Figura 17.2. *Shortcut*-ul pentru lansarea instrumentului `Help Workshop`

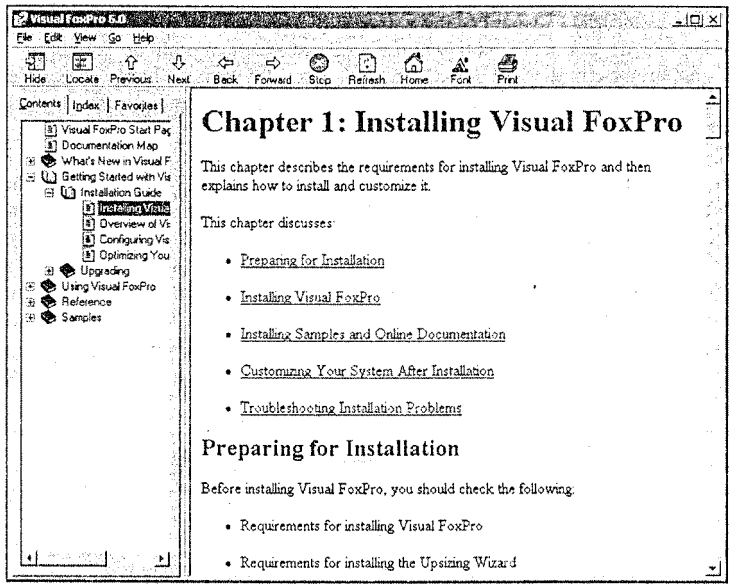


Figura 17.3. Asistenţă on-line în noul format `HTML Help`

Instrumentul pentru realizarea de documentaţii în acest nou format este numit `HTML Help Workshop`. Deşi nu se instalează implicit o dată cu pachetul Visual Studio, kit-ul de instalare se găseşte pe primul CD al acestuia, având numele `HTMLHELP.EXE`.

Ultima versiune poate fi obţinută şi de pe site-ul firmei Microsoft (figura 17.4).

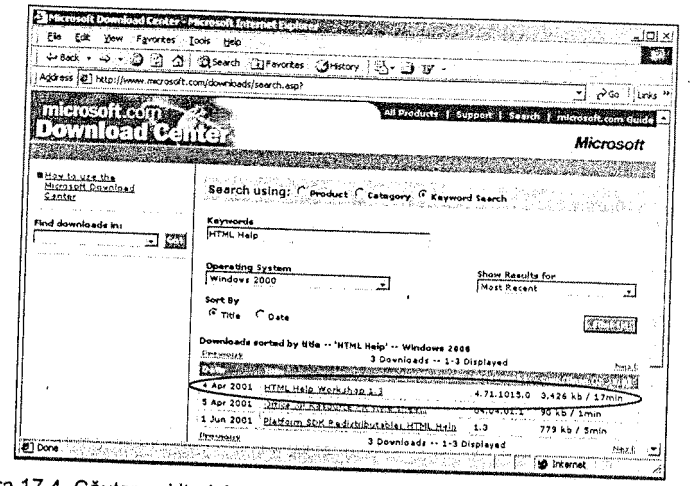


Figura 17.4. Căutarea kit-ului `HTML Help Workshop` pe site-ul `microsoft.com`

Fişierele *help* generate cu *HTML Help Workshop* pot prezenta următoarele caracteristici:

- pagini de cuprins (`contents`), cu vizualizare ierarhică a subiectelor (`topics`);
- index bazat pe cuvinte-cheie, stabilite la proiectarea fişierelor cu informaţii de ajutor;
- capacitate opţională de căutare (`full-text search`) după expresii furnizate de utilizator;
- text cu diferite fonturi, culori, animaţie;
- imagini statice (formatele `.wmf`, `.gif`, `.jpg`, `.png`[75]);
- imagini animate (`.gif` animate, fişiere `.avi`[76]), sunete;
- includere de scripturi (JavaScript, VBScript), asemenea oricărei pagini Web;
- posibilitatea navigării (`hyperlink`) la alt subiect `help` sau la o adresă Internet;
- personalizarea meniului şi barei de instrumente la momentul afişării etc.

Modalităţile în care utilizatorul unei aplicaţii poate obţine informaţii pe parcursul derulării acesteia sunt cele obişnuite în mediul Windows:

- ajutor general, cu posibilitate de navigare, prin meniul `Help` al aplicaţiei de care „se leagă" *help*-ul (sau prin tasta `F1`);
- acţionarea unui buton special destinat, afişat într-o fereastră (poate filtra informaţiile de ajutor pentru explicaţii în situaţia respectivă);
- ajutor dependent de context (prin apăsarea tastei `F1`, printr-o opţiune adecvată într-un meniu contextual etc.) – poate filtra informaţiile de ajutor pentru explicaţii detaliate privind situaţia respectivă;
- ajutor de tipul `what's this?`, afişat printr-un buton special [?], prezent în bara de titlu a unor ferestre, ce poate afişa informaţii precise, scurte, despre un anumit element indicat cu mouse-ul după ce s-a apăsat butonul special.

75. În ordine, *Windows MetaFile* (grafică vectorială), *Graphics Interchange Format*, *Joint Picture Experts Group*, *Plasma Network Graphics* – formate grafice uzuale în paginile Web.
76. *Audio Video Interleave* – format pentru animaţie (inclusiv filme).

HTML Help Workshop are şi capacitatea de a converti fişiere `help` din formatul vechi în cel nou, tip `HTML`.

17.2. Cum se utilizează HTML Help Workshop ?

La lansarea programului HTML Help Workshop, pe ecran se afişează fereastra din figura 17.5. Primul pas în realizarea unui sistem de ajutor interactiv este crearea unui *proiect*, care joacă rolul de *catalog* pentru fişierele de diferite categorii ce contribuie la crearea produsului final. Operaţia se lansează prin intermediul meniului `File→New`. Se va afişa o fereastră cu câteva opţiuni, din care se alege `Project`.

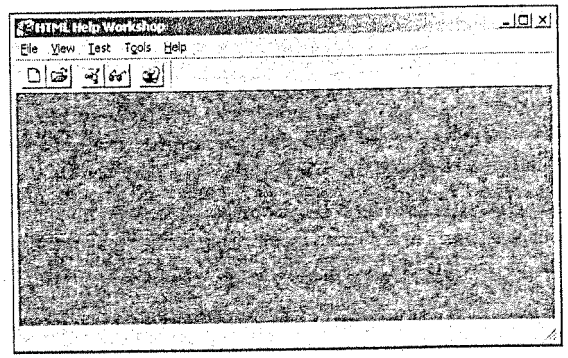


Figura 17.5. Aspectul ecranului la lansarea aplicaţiei `HTML Help Workshop`

Pot exista următoarele categorii de fişiere necesare pentru a crea un sistem de ajutor pentru o aplicaţie:

- fişierul-proiect (are extensia `.hhp`) – este fişierul principal;
- fişiere cu subiecte (`topic files`): fişiere `HTML` care descriu în amănunt fiecare modul al aplicaţiei pentru care se doreşte documentarea. Pot cuprinde elemente de grafică şi legături spre alte fişiere (`HTML` sau non-`HTML`), porţiuni de fişiere `HTML` (indicate cu `bookmark`-uri) sau spre locaţii pe Web;
- fişiere grafice, multimedia etc., în cazul în care se intenţionează apelarea lor din fişierele de subiecte;
- fişiere tip cuprins sau tablă de materii (`contents`), care organizează subiectele pe categorii; sunt facultative şi au extensia `.hhc`;
- fişiere tip index, care permit afişarea ordonată a subiectelor şi localizarea lor după cuvinte-cheie; sunt facultative; au extensia `.hhk`.

După realizarea şi înglobarea într-un proiect a acestor categorii de fişiere, acesta se compilează, obţinându-se un nou fişier cu extensia `.chm`. Acesta este singurul care trebuie distribuit împreună cu aplicaţia pentru care a fost creat.

Notă. Dacă fişierele de subiecte apelează programe externe ori alt fel de fişiere (sunete, animaţii etc.), vor trebui distribuite şi acestea.

Să revenim la situaţia în care tocmai am optat pentru crearea unui nou fişier-proiect. Un `wizard` (asistent) ne va propune stabilirea unui nume pentru proiect (figura 17.6). Dialogul surprins în figura 17.7 propune ataşarea unor fişiere de tip subiect, conţinut sau index, dacă au fost create în prealabil.

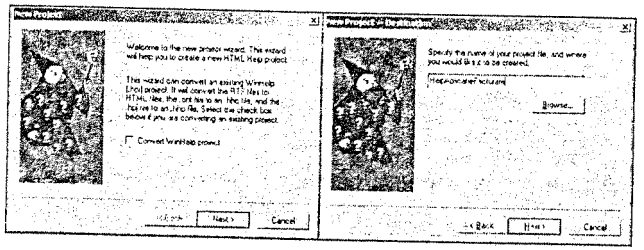


Figura 17.6. Precizarea unui nume de proiect în aplicaţia `HTML Help Workshop`

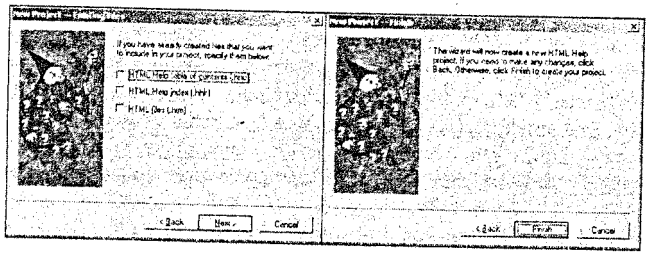


Figura 17.7. Opţiunea pentru includerea unor fişiere deja create

Dacă nu există fişiere `HTML` ori din celelalte categorii, acestea vor putea fi inserate ulterior.

17.2.1. Crearea fişierelor de subiecte (conţinut)

Înainte de a crea fişierele de subiecte, este recomandabilă realizarea unei schiţe („hartă") a aplicaţiei ce va fi documentată. Spre exemplu, se poate ţine seama de opţiunile importante din meniul acesteia. Se creează câte un fişier `HTML` pentru fiecare subiect (aşa cum vom proceda în continuare) sau un fişier comun mai multor subiecte.

Crearea unui nou fişier de subiecte se realizează prin opţiunea `File→New...` (vezi figura 17.8).

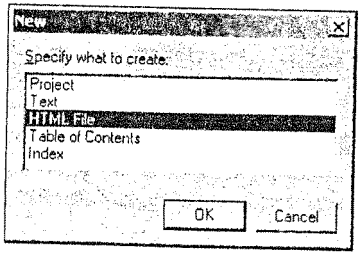


Figura 17.8. Crearea unui fişier de subiecte (`HTML`)

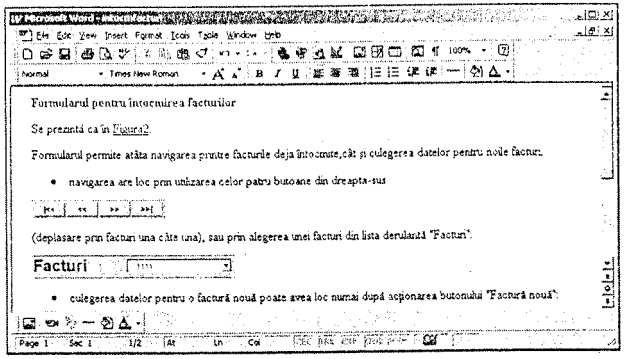


Figura 17.9. Microsoft Word ca editor HTML

De remarcat că editorul HTML încorporat lucrează în mod text, necesitând câteva cunoştinţe privind limbajul HTML (informaţii de ajutor sunt oferite chiar în Help→HTML Tag Reference). Se poate însă folosi orice instrument extern care poate salva documente în format HTML; majoritatea instrumentelor dedicate HTML permit lucrul în manieră vizuală. Exemplele de faţă au fost realizate cu Microsoft Word 97 SR-1[77] (*patch*-ul SR-1 se poate obţine prin download de la http://office.microsoft.com/Assistance/9798/sr1howtoget.aspx). Varianta fără SR-1 are unele probleme la salvarea în format HTML.

Întrucât prezentul material are un rol demonstrativ şi nu se intenţionează crearea unui sistem de ajutor complet pentru o aplicaţie, au fost ataşate doar trei fişiere HTML:

- unul care foloseşte şi ca pagină de prezentare a aplicaţiei[78] (startpage.htm);
- unul pentru informaţii cu privire la formularul de consultare a facturilor (vezi capitolul 10);
- unul pentru informaţii cu privire la formularul de întocmire a facturilor (vezi capitolul 10).

Ultimele două fişiere cuprind şi imagini ale formularelor în curs de execuţie, obţinute prin aşa-numitele „capturi de ecran". Pentru acestea trebuie folosit un program special. Livrat şi instalat o dată cu HTML Help Workshop, programul HTML Help Image Editor poate fi apelat cu opţiunea Tools→HTML Help Image Editor sau separat, din meniul Start→Programs.... Acest accesoriu permite, printre diferite manipulări şi conversii de fişiere grafice, şi captarea conţinutului ecranului (Alt+K, apoi deplasare în fereastra de „fotografiat" şi apăsarea tastei F11; pentru copierea întregului ecran, se apasă F12).

HTML Help Image Editor este un instrument performant, dar nu indispensabil. Astfel, se pot utiliza tastele PrintScreen (captează întregul ecran) ori Alt+Print Screen (captează fereastra curentă). Ulterior acestei operaţii, se lansează un program de grafică precum *Paint* şi se execută comanda Edit→Paste. În loc de Paint se poate folosi, cu rezultate mai bune (recunoaşte mai multe formate grafice), *Microsoft Photo Editor* (inclus în pachetul Office 97). Acest din urmă program are opţiunea Edit→Paste As New Image, prin care creează o imagine exact cât captura de mare (spre deosebire de imaginea

77. Service Release 1. Între timp, este disponibil şi SR-2.
78. Am convenit ca aplicaţia să se numească „Facturare".

din Paint, care are dimensiuni prestabilite), precum şi o serie de comenzi ce permit copieri ori decupări ale unor porţiuni de imagine.

Notă. Copierea ecranelor cu (Alt) PrintScreen nu captează şi săgeata cursorului mouse-ului. HTML Help Image Editor are opţiune specială pentru aceasta (Capture→Preferences→General).

Să presupunem că am terminat cele trei fişiere în format HTML. Vom reveni la proiectul nostru, în care pentru moment nu s-a adăugat nimic şi care ar trebui să arate ca în figura 17.10.

Observăm că, de-a lungul laturii stângi a ferestrei, se găsesc câteva butoane, ale căror roluri le explicăm în continuare (de sus în jos):

(1) opţiunile proiectului;
(2) adăugare/ştergere de fişiere HTML cu subiecte;
(3) definiţii de ferestre personalizate pentru afişarea conţinutului;
(4) intrumente pentru interconectarea şi afişarea informaţiilor de ajutor (grupate sub numele *HTML Help API*);
(5) vizualizarea fişierelor de conţinut, în cod-sursă HTML;
(6) salvarea fişierului-proiect;
(7) salvarea şi compilarea fişierului-proiect (se obţine fişierul .chm).

Printre opţiunile proiectului se găsesc şi cele privitoare la limba în care sunt redactate fişierele de ajutor, precum şi la fontul utilizat, de care trebuie să se ţină cont la afişarea în browser (aţi încărcat vreodată o pagină în rusă sau japoneză fără a avea suportul multilingual instalat în sistemul de operare Windows?). În exemplul nostru, am optat pentru limba română, efectuând un click pe butonul (1). Pasul următor este redat prin intermediul figurii 17.11.

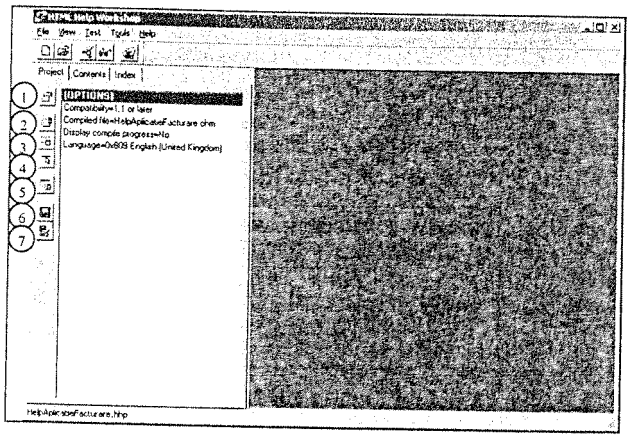


Figura 17.10. Proiect HTML Help Workshop fără fişiere ataşate

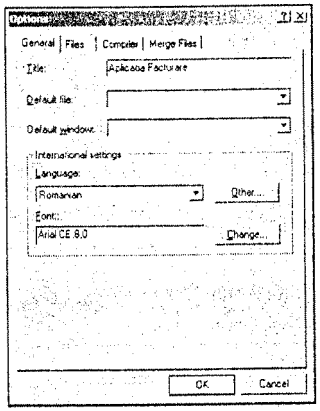


Figura 17.11. Stabilirea opțiunilor privind limba și setul de caractere

Tot din acest dialog se poate stabili și un titlu pentru fereastra în care se va vizualiza documentația, în forma sa finală (`Title`).

Adăugarea fișierelor cu subiecte se face prin acționarea butonului (2), `Add/Remove topic files` (vezi figura 17.12). După adăugare, unul dintre respectivele fișiere poate fi declarat ca pagină principală a sistemului `Help`. Aceasta va fi afișată întotdeauna automat o dată cu lansarea sistemului de ajutor sau dacă s-a căutat un subiect inexistent (vezi mai jos paragrafele *Crearea unui cuprins* și *Crearea unui index*). Pentru aceasta, este necesar să ne întoarcem la fereastra de dialog din figura 17.11, caseta `Default file`, unde s-a ales fișierul `startpage.htm`.

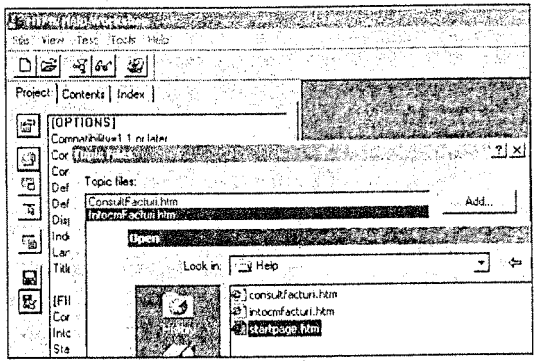


Figura 17.12. Adăugarea de fișiere cu subiecte

În acest moment, proiectul cuprinde trei fișiere HTML: `startpage.htm`, `intocmfacturi.htm` și `consultfacturi.htm`. Putem recurge la o compilare „de probă" a proiectului (butonul (7)). Va rezulta un fișier `.chm` funcțional (pentru vizualizare,

se acționează butonul [79]), dar care va afișa întotdeauna doar pagina declarată ca principală (în acest caz, `startpage.htm`).

17.2.2. Crearea unui cuprins (*Table of Contents*)

Cum se procedează pentru a accesa diferitele capitole (fișiere) ale documentației? Ei bine, este nevoie de proiectarea unei „table de materii" (cuprins) pentru sistemul nostru de ajutor. În acest scop, se apelează comanda `File→New...` și opțiunea `Table of Contents`. Se va crea un fișier fără titlu, pe care ulterior îl putem salva cu orice nume dorim (păstrând însă extensia `.hhc`). Imediat, vom putea trece în pagina `Contents` a ferestrei *HTML Help Workshop*. De-a lungul laturii stângi a ferestrei se găsesc câteva butoane (vezi figura 17.13), ale căror roluri le explicăm în continuare (de sus în jos):

(1) opțiunile cuprinsului;
(2) adăugarea de anteturi (informații grafice și textuale cu rol de grupare a subiectelor);
(3) adăugarea de pagini (link-uri către anumite fișiere de subiecte);
(4) modificarea paginii/antetului selectat(e);
(5) ștergerea paginii/antetului selectat(e);
(6) mutarea paginii/antetului selectat(e) mai sus în ierarhie;
(7) mutarea paginii/antetului selectat(e) mai jos în ierarhie;
(8) mutarea paginii/antetului selectat(e) spre dreapta (un nivel „mai adânc") în ierarhie;
(9) mutarea paginii/antetului selectat(e) spre stânga (un nivel „mai general") în ierarhie;
(10) afișarea sursei HTML a fișierului de subiecte corespunzător paginii selectate;
(11) salvarea fișierului de cuprins.

Vom crea pentru început o pagină cu butonul (2), intitulată `Despre "Facturare"`, iar cu butoanele `Add...`, respectiv `Browse...` (vezi figura 17.13, unde deja au fost adăugate câteva pagini) vom parcurge structura de directoare până la fișierul `StartPage.htm`.

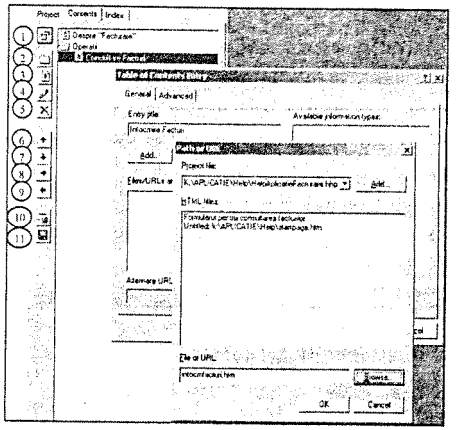


Figura 17.13. Adăugarea la cuprins a unei referințe de pagină

79. Altă modalitate: în Windows Explorer, se caută fișierul cu același nume ca proiectul, dar cu extensia `.chm`, și se lansează prin dublu-click.

În continuare, se adaugă un antet – cu butonul (1) – intitulat `Operatii`. Anteturile se vor afişa cu pictogramele unor foldere şi nu trebuie să facă apel la vreo pagină `HTML`.

Sub antetul `Operatii` se adaugă două pagini (cu butonul (3)):

- `Consultare facturi;`
- `Intocmire facturi;`

Lor le corespund documentele `ConsultFacturi.htm`, respectiv `Intocm Facturi.htm`. Asocierea se realizează după cum s-a explicat mai sus (vezi şi figura 17.13). Ultimul element pe care-l mai adăugăm în această tablă de materii experimentală este un antet intitulat `Liste`, căruia nu-i vom asocia deocamdată nici o pagină (din acest motiv, în fereastra produsului final va fi afişat el însuşi ca o pagină).

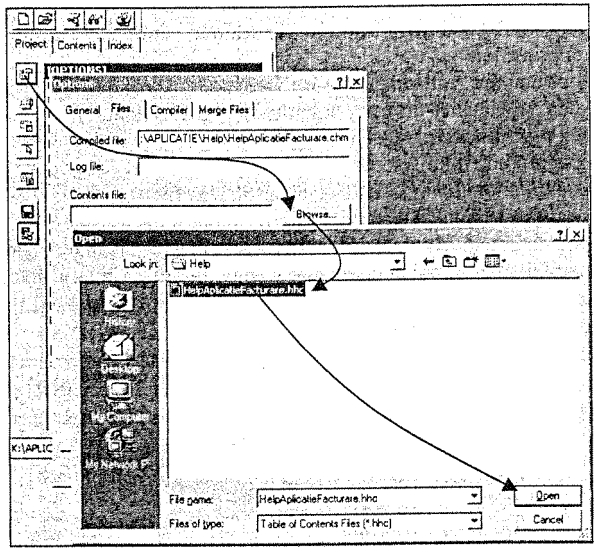


Figura 17.14. Adăugarea fişierului de cuprins la proiect

Dacă veţi efectua încă o compilare a proiectului, urmată de vizualizarea fişierului final, ar trebui să apară, în stânga ferestrei de vizualizare, cadrul de pagină `Contents` (figura 17.15). După cum se observă, antetele sunt afişate ca foldere în manieră Windows Explorer, cu posibilitatea expandării/condensării.

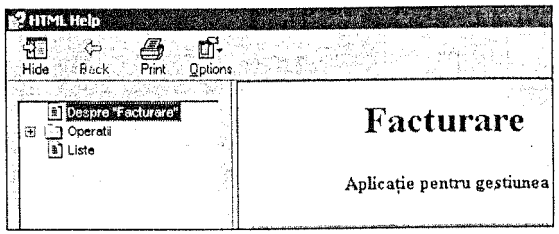


Figura 17.15. Sistem de *help* care posedă cuprins (faza de execuţie)

Notă. Se poate asocia acelaşi document `HTML` oricâtor pagini din cuprins.
Nu se va omite adăugarea fişierului de cuprins în proiect (figura 17.14).

Observaţie. În timpul compilării proiectului de *help*, urmăriţi mesajele afişate în dreapta ferestrei programului *HTML Help Workshop* (se cere şi specificarea unei căi şi a unui nume pentru un fişier-jurnal – `.log`, în care se scriu mesajele compilatorului). Un mesaj precum cel din figura 17.16 semnifică faptul că fişierul compilat este folosit undeva (verificaţi dacă nu e deschis sau dacă nu e deschisă o altă aplicaţie ce face referire la el).

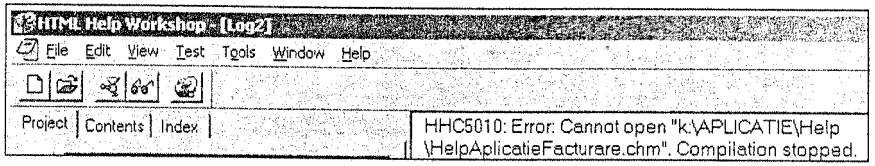


Figura 17.16. Proiectul nu poate fi compilat

17.2.3. Crearea unui index

O documentaţie voluminoasă se parcurge mult mai uşor dacă subiectele sale pot fi identificate prin unul sau mai multe cuvinte-cheie, stabilite încă în etapa de proiectare a sistemului de ajutor. Programul HTML Help Workshop utilizează în acest scop fişierele index (au extensia `.hhk`).

Crearea fişierului index are loc în mod asemănător cu realizarea fişierului de cuprins prezentată în paragraful anterior, şi anume prin `File→New... Index`, ori pur şi simplu prin selecţia paginii `Index` din partea stângă a ferestrei programului HTML Help Workshop. Fişierul se poate salva cu orice nume, importantă fiind extensia `.hhk`. Un fişier index se completează urmând indicaţiile din figura 17.17. De-a lungul laturii stângi a ferestrei se găsesc câteva butoane, ale căror roluri le explicăm în continuare (de sus în jos):

(1) proprietăţile indexului;
(2) inserarea unui cuvânt-cheie;
(3) modificarea cuvântului-cheie selectat;
(4) ştergerea cuvântului-cheie selectat;
(5) mutarea cuvântului-cheie selectat mai sus în ierarhie;
(6) mutarea cuvântului-cheie selectat mai jos în ierarhie;
(7) mutarea cuvântului-cheie selectat spre dreapta (un nivel „mai adânc") în ierarhie;
(8) mutarea cuvântului-cheie selectat spre stânga (un nivel „mai general") în ierarhie;
(9) sortarea cuvintelor-cheie ce corespund aceleiaşi selecţii;
(10) vizualizarea sursei `HTML` a fişierului-subiect referit de cuvântul-cheie selectat (echivalent cu dublu-click pe cuvântul-cheie selectat);
(11) salvarea fişierului index.

Cuvintele-cheie se organizează pe cel puţin două niveluri: principal (general) şi secundar (detaliat). De fapt, numărul de paliere depinde de gradul de detaliu urmărit pentru explicaţiile furnizate. Pentru „promovarea", respectiv „retrogradarea" unui cuvânt-cheie se utilizează butoanele (7) şi (8). Atenţie la folosirea butoanelor (5) şi (8): dacă selecţia este un cuvânt-cheie de nivel general, el va fi mutat împreună cu toate cuvintele-cheie secundare

care îi aparţin (sunt afişate sub el şi mai la dreapta). În plus, aceste cuvinte-cheie secundare se pot sorta în ordine alfabetică, folosind butonul (10).

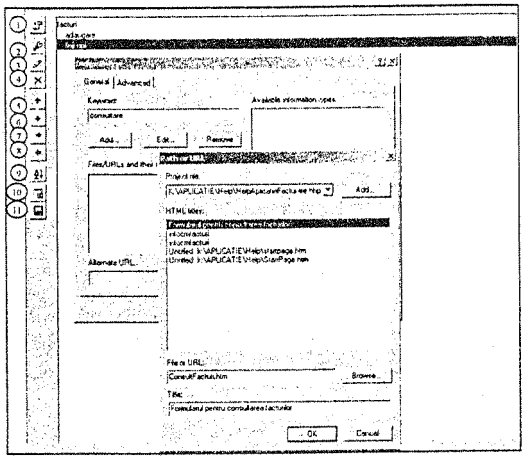


Figura 17.17. Adăugarea de cuvinte-cheie şi referinţe în fişierul index

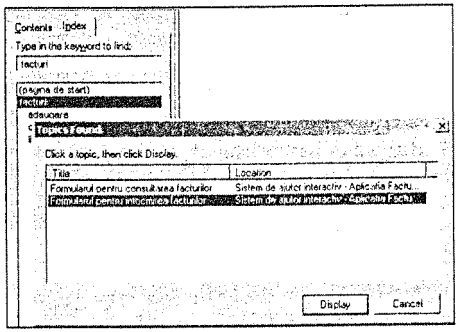


Figura 17.18. Mai multe subiecte asociate aceluiaşi cuvânt-cheie (faza de execuţie)

Notă. Se poate asocia acelaşi document `HTML` oricător cuvinte-cheie din fişierul index şi invers. În această manieră se procedează pentru cuvintele-cheie principale: se scriu referinţe la toate documentele `HTML` la care se vor feri cuvintele-cheie secundare. În cazul de faţă, cuvântului-cheie `facturi` i s-au asociat toate fişierele `HTML` care se referă la operaţii asupra facturilor (întocmire, consultare etc.). În sistemul de ajutor final, utilizatorul va avea posibilitatea să aleagă un subiect din mai multe posibile asociate aceleiaşi opţiuni (figura 17.18).
Nu se va omite adăugarea fişierului index în proiect (figura 17.19).

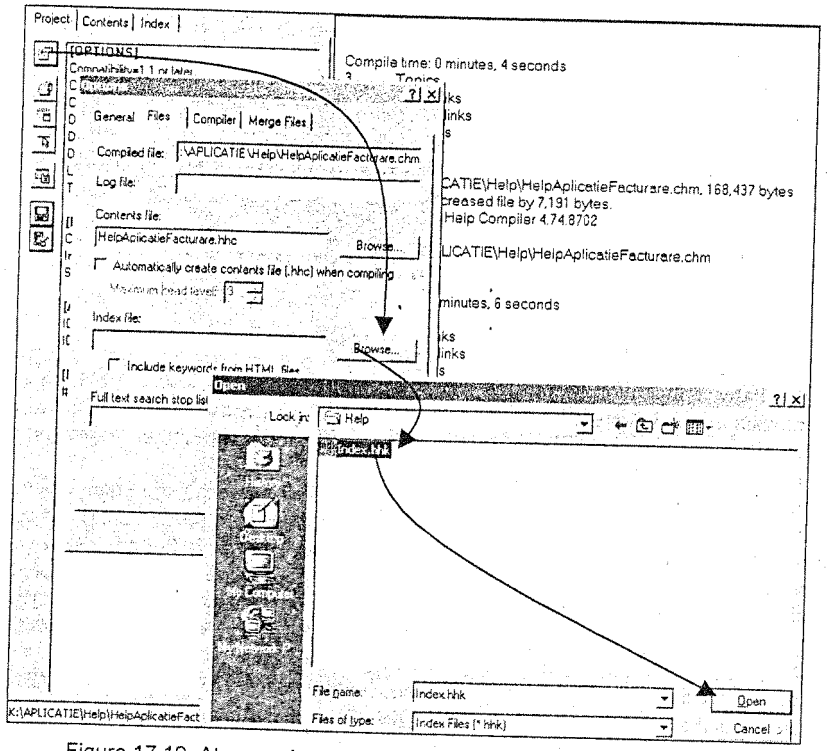


Figura 17.19. Ataşarea la proiectul sistemului *help* a unui fişier tip index

După această operaţiune se va compila din nou proiectul şi se va încerca vizualizarea produsului final. Dacă lucrurile au decurs fără probleme, fereastra de afişare ar trebui să aibă acum, în partea stângă, două cadre de pagină: `Contents`, respectiv `Index` (vezi figura 17.18).

17.2.4. Ataşarea fişierului de *help* compilat la o aplicaţie Visual FoxPro

Ataşarea fişierului compilat la o aplicaţie Visual FoxPro poate fi considerată o misiune destul de uşoară. Orice proiect Visual FoxPro care va fi transformat într-un program executabil dispune de o componentă principală, fie ea `.prg`, `.mpr` (meniu) ori formular `.scx`. Pentru ca sistemul de ajutor să fie disponibil pe parcursul întregii aplicaţii, în programul principal trebuie inclusă comanda:

```
SET HELP TO <nume-fişier.chm>
```

Ulterior, apelarea informaţiilor de ajutor are loc printr-o simplă comandă `HELP`. Aceasta va lansa în execuţie fişierul specificat, cu afişarea paginii implicite. De exemplu, comanda `HELP` se poate plasa în meniul aplicaţiei. O altă soluţie este scrierea unei comenzi `ON KEY LABEL F1 HELP`, astfel încât sistemul de ajutor să poată fi apelat prin apăsarea tastei `F1`. Există şi varianta comenzii `HELP <cuvânt-cheie>`, în cazul nostru `HELP`

facturi, pentru afişarea imediată a paginii HTML care a fost asociată cu respectivul cuvânt-cheie.

Nu trebuie omis ca, în modulul de program ce se va executa la ieşirea din aplicaţie (acolo unde de obicei are loc închiderea bazelor de date şi tabelelor, restaurări ale unor parametri de mediu etc.), să se plaseze o comandă SET HELP TO, care are rolul restabilirii sistemului de *help* la fişierul .chm implicit pentru Visual FoxPro. Dacă este cazul, se va folosi şi comanda ON KEY, pentru a degreva tasta F1 de sarcina afişării informaţiilor de ajutor.

Este bine să efectuăm astfel de teste pentru început în varianta de lucru (necompilată) a proiectului .pjx, eventual chiar în fereastra de comenzi.

17.2.5. Asistenţă „senzitivă la context"

Majoritatea aplicaţiilor ce rulează sub Windows, atunci când este apăsată tasta F1 (generalizată ca modalitate de apelare rapidă a sistemului de asistenţă), reacţionează prin afişarea diferitor porţiuni ale documentaţiei, corespunzătoare situaţiei de pe ecran (practic, poziţiei în care se află cursorul).

Într-un mediu de dezvoltare vizual, aşa cum este Visual FoxPro, majoritatea obiectelor ce pot fi inserate într-un formular dispun de proprietatea HelpContextID. Această proprietate are o valoare implicită ce nu semnifică nimic (în Visual FoxPro, valoarea 0). Programatorul poate atribui însă diferite valori numerice acestei proprietăţi. Asistenţa senzitivă la context va funcţiona în cazul în care valorile stabilite pentru proprietatea HelpContextID în cazul diferitelor obiecte ale aplicaţiei corespund unor referinţe special definite în proiectul realizat cu HTML Help Workshop.

Aceste referinţe sunt nişte numere (pozitive) care desemnează anumite pagini HTML din fişierele cu subiecte. În cadrul proiectului help, aceste numere trebuie să fie unice (reciproca nu este valabilă, întrucât un singur fişier HTML poate fi referit de mai multe numere distincte). Fiecărui număr (greu de ţinut minte) i se asociază însă o combinaţie de caractere sugestivă (mnemonică); prin convenţie, aceste combinaţii încep cu caracterele IDH_. Descrierea lor trebuie să se facă într-un fişier distinct, de tip ASCII, care are neapărat extensia .h, după tipicul fişierelor *header* din C, C++ (sau din Visual FoxPro), şi care conţine doar directive #DEFINE. Acest fişier este cunoscut sub denumirea de *fişier MAP* (engl. *map* = hartă). În cazul nostru (proiectul HelpAplicatie Facturare.hhp), fişierul AplicatieFacturare.h conţine doar următoarele linii:

```
#DEFINE IDH_ConsultFacturi 1 //subiectul formularului "Consultare facturi"
#DEFINE IDH_IntocmFacturi 2 //subiectul formularului "Intocmire facturi"
```

Cele două caractere „//" succesive denotă un comentariu, deci compilatorul le va ignora. Nu se face distincţie între literele mici şi cele mari.

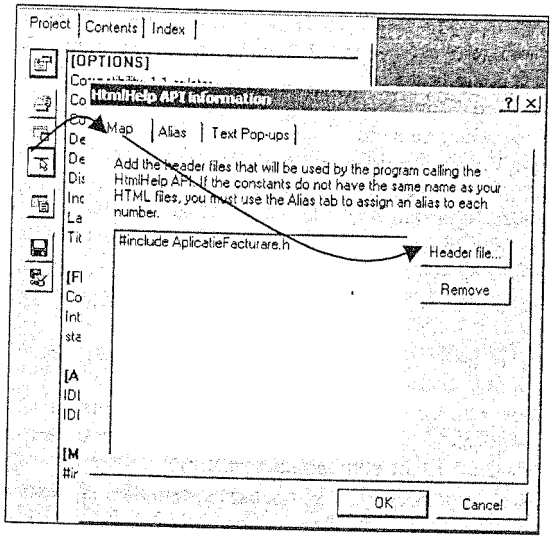


Figura 17.20. Ataşarea unui fişier MAP

După ce se realizează un astfel de fişier de „referinţe la context", el trebuie ataşat proiectului *help* (ulterior ataşării, se pot totuşi adăuga oricâte directive #DEFINE sunt necesare, dar proiectul va trebui recompilat de fiecare dată). Ataşarea fişierului MAP se face urmând instrucţiunile din figura 17.20.

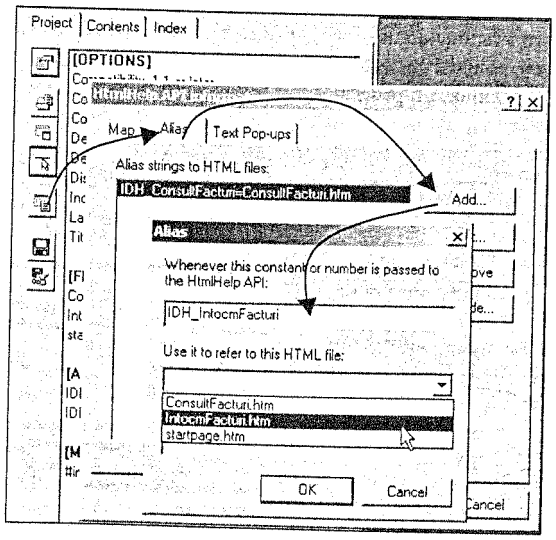


Figura 17.21. Stabilirea referinţelor la fişiere HTML

În continuare, se utilizează cadrul de pagină `Alias`, spre a întreprinde o asociere între constantele din fişierul `MAP` şi fişierele `HTML` cu subiecte (vezi figura 17.21).

Urmează o nouă compilare a proiectului de asistenţă. Testarea modului de funcţionare în Visual FoxPro se face, în linii mari, astfel:

- se deschide o sesiune de lucru Visual FoxPro;
- în fereastra de comenzi, se execută comanda:

```
SET HELP TO "k:\Aplicatie\Help\HelpAplicatieFacturare.chm"
```

(sau alt nume de fişier, după caz);

- se scrie şi execută următoarea comandă: `HELP ID 1`.

Ar trebui să apară, în fereastra de vizualizare a informaţiilor de ajutor, pagina `HTML` asociată anterior numărului `1`, respectiv mnemonicei `IDH_ConsultFacturi`.

Notă. Uneori, fereastra de asistenţă se deschide minimizată, dar clipeşte pe *taskbar*.

Dacă testul a reuşit, urmează cuplarea propriu-zisă a „contextului" la formularele din aplicaţia Visual FoxPro. Vom începe cu formularul de consultare a facturilor, numit `frmTree_List.scx`. Paşii de parcurs sunt următorii:

- în metoda `Init` a formularului, se ataşează sistemul de asistenţă on-line:
  `SET HELP TO "\Aplicatie\Help\HelpAplicatieFacturare.chm"`
- în procedura evenimentului `Destroy`, scriem:
  `SET HELP TO`
- pe formular se plasează un buton, cu proprietăţile:
  - `Caption: Help`
  - `HelpContextID: 1`
  - `Name: cmdHelp`
- în procedura evenimentului `Click` pentru butonul `cmdHelp`, vom invoca sistemul de ajutor prin comanda: `HELP ID THIS.HelpContextID`;
- se testează formularul prin lansare în execuţie şi click pe butonul nou-creat (figura 17.22).

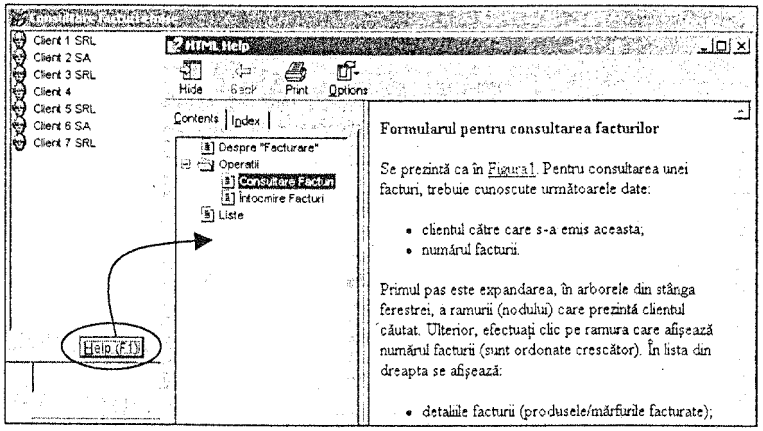


Figura 17.22. Informaţii de ajutor în funcţie de context

Paşii de mai sus trebuie parcurşi pentru orice formular din proiectul Visual FoxPro, cu precizarea că se va schimba valoarea proprietăţii `HelpContextID` de la un formular la altul.

17.3. Construirea kit-ului de distribuţie

Livrarea aplicaţiei către utilizatorii finali presupune existenţa unui kit de distribuţie. Acesta va conţine o formă compilată a codurilor-sursă (programe, formulare, rapoarte, meniuri), baza de date şi alte tipuri de fişiere necesare funcţionării aplicaţiei (imagini, sunet, fişiere text de configurare etc.). Realizarea sa presupune parcurgerea a doi paşi:

- compilarea codurilor-sursă în unul sau mai multe fişiere de tip `.exe` sau `.dll`;
- crearea fişierelor de instalare (`setup`).

În funcţie de modul de abordare în proiectarea arhitecturii generale, produsul finit poate îmbrăca două forme:

- forma **monolitică** – diferitele componente (proceduri, formulare, rapoarte etc.) aparţin unui singur proiect, caz în care obţinem un singur fişier final executabil care le înglobează pe toate;
- forma **modulată** – aplicaţia este organizată în *module funcţionale* distincte implementate fizic prin intermediul claselor publice care vor fi compilate şi livrate sub formă de componente COM[80].

Nu încape nici o îndoială că o aplicaţie modulată va fi mult mai uşor de întreţinut atât timp cât eventualele modificări sau actualizări ce apar după implementare vor avea implicaţii doar asupra unuia sau câtorva dintre module.

17.3.1. Noţiunea de „componentă principală" a unui proiect

Pentru a putea fi compilat, orice proiect trebuie să dispună de o *componentă principală*. Aceasta este de fapt un program, un formular sau un meniu pentru care va fi selectată opţiunea `Project→Set Main` din meniul sistem şi care va servi drept *punct de plecare pentru aplicaţia compilată*. În general, o asemenea componentă (listingul 17.1) trebuie să asigure îndeplinirea următoarelor obiective:

- configurarea parametrilor de mediu (căile implicite şi alternative de căutare, formatul datei calendaristice, parametrii interni de securitate etc.) şi deschiderea bazei de date;
- iniţializarea şi afişarea interfeţei aplicaţiei (meniu sau formular);
- iniţierea *procesării evenimentelor* – comanda `READ EVENTS`.

Din momentul execuţiei comenzii `READ EVENTS`, programul principal este întrerupt, iar Visual FoxPro intră în bucla de procesare a evenimentelor (un gen de *standby*) aşteptând intervenţiile utilizatorului: selecţia unei opţiuni din meniu sau declanşarea unor evenimente recunoscute de obiectele formularelor (click, dublu-click etc.).

Pentru a întrerupe bucla, vom utiliza comanda `CLEAR EVENTS`.

Atenţie! Lipsa comenzii `READ EVENTS` din componenta principală a unei aplicaţii compilate va determina o apariţie „meteorică" a interfeţei pe ecran, după care aceasta se va închide.

80. Vezi capitolul 14, pentru detalii asupra componentelor COM.

Listing 17.1. Conținutul standard al componentei principale

```
SET SAFETY OFF
SET CENTURY ON
SET EXACT ON
SET DATE TO british
ON SHUTDOWN DO prgiesire   && pentru functionarea butonului de inchidere a ferestrei pricipale
DO FORM frmlogon      && un formular de autentificare a utilizatorului
** sys(16)- numele (cu tot cu calea intreaga) a fisierului care se executa (fisierul curent)
** spre exemplu : h:\aplicatie\principalu.prg
x=RAT("\",SYS(16))    && pozitia ultimului "\" in sirul returnat de sys(16)

zona_lucru_curenta=" ' "+LEFT(SYS(16),x-1)+" ' "
** numele directorului in care se executa fisierul a fost incadrat intre apostroafe ( " ' " ) : daca unul din
** directoarele ce compun calea contine spatii, comanda trebuie sa arate astfel:
** set defa to 'c:\program files\vinzari'

SET DEFAULT TO &zona_lucru_curenta
SET PATH TO database,help,imagini,imagini\icons,meniuri,progs,reports,forms,help
SET HELP TO HelpAplicatieFacturare.chm
ON KEY LABEL F1 HELP
OPEN DATABASE vinzari SHARED
_SCREEN.ICON='green16.ico'
_SCREEN.PICTURE='background.jpg'
_SCREEN.CAPTION='Evidenta clienti'
_SCREEN.WINDOWSTATE=2    && fereastra principala va fi implicit maximizata
DO initializare
DO mainmenu.mpr                  && lansarea meniului principal

READ EVENTS
```

În listingul 17.1 observăm următoarele:

- la stabilirea căilor alternative de căutare (comanda `SET PATH`) au fost luate în considerare şi subdirectoare ce nu vor fi livrate utilizatorului final (ex.: meniuri, rapoarte etc.). Acestea sunt însă necesare în faza de testare, înainte de compilarea aplicației, pentru buna funcționare a modulelor. Oricum, orice director ce nu va fi regăsit fizic pe disc va fi ignorat, fără ca Visual FoxPro să genereze vreun mesaj de eroare;
- utilizarea funcției `SYS(16)` pentru a obține calea spre directorul-rădăcină al aplicației, oricare ar fi poziția fizică pe disc a acestuia. Pentru a putea fi utilizată această funcție, componenta principală, ca şi fişierul executabil final trebuie să se găsească în directorul-rădăcină al aplicației;
- se utilizează ghilimele pentru a încadra calea implicită de căutare în cazurile în care aceasta conține spații;
- lansăm în execuție procedura generalizată de deschidere a tabelelor bazei de date (în mod partajat, dacă lucrăm în rețea – listingul 17.2);
- `ON SHUTDOWN` – instrucțiune cu ajutorul căreia specificăm procedura ce se va executa atunci când utilizatorul închide aplicația din butonul specific oricărei ferestre Windows;
- `_SCREEN` – variabilă sistem de tip obiect prin intermediul căreia accesăm proprietățile şi metodele ferestrei principale Visual FoxPro în scopul personalizării interfeței aplicației.

Listing 17.2. Exemplu de cod generalizat pentru deschiderea tabelelor unei baze de date

```
                    ********procedura INITIALIZARE**********
*functia ADBOBJECTS populeaza un vector cu obiectele bazei de date curente
ADBOBJECTS(vTabele, "TABLE")
FOR i= 1 TO ALEN(vTabele,1)
    USE (vTabele(i)) IN 0 SHARED
ENDFOR
```

Conținutul procedurii de închidere a aplicației îl regăsim în listingul 17.3:

Listing 17.3. Procedura `Prgiesire` – se execută la ieşirea din aplicație

```
CLOSE DATA ALL
SET HELP TO
ON KEY
DO COMPACTARE  && procedura de ştergere definitivă a înregistrărilor marcate pentru ştergere
CLEAR EVENTS
QUIT
```

Atunci când lucrăm în rețea (aceeaşi bază de date este partajată între mai mulți utilizatori), comanda `PACK` nu poate fi utilizată în interiorul aplicației. De aceea trebuie concepută o procedură adecvată care:

- încearcă să deschidă baza de date în mod exclusiv;
- execută comanda `PACK` pe tabele dacă deschiderea exclusivă a reuşit.

Listing 17.4. Procedura de compactare a tabelelor bazei de date (`PACK`)

```
a=0
ON ERROR a=ERROR()   && instituie o tratare personalizata a erorilor
* se incerca deschiderea exclusiva a bazei de date
OPEN DATABASE Vinzari EXCLUSIVE
IF a=1705       && s-a semnalat eroarea "file access denied" (baza este deschisă de alt utilizator)
    ON ERROR
    RELEASE a            && sterge variabila din memorie si elibereaza zona respectiva
    RETURN
ENDIF
DIMENSION aTabele(1)
LOCAL i
ON ERROR a=ERROR()
WAIT WIND "Se elimina datele sterse, aveti putintica rabdare..." NOWAIT

* ne asiguram ca inregistrarile marcate
* pentru stergere sunt vizibile
SET DELETED OFF
=ADBOBJECTS(aTabele, "TABLE") && aflam care sunt tabelele
FOR i=1 TO ALEN(aTabele,1)
    tab_ = ALLTRIM(aTabele(i))
    USE (tab_) IN 0 EXCLUSIVE
    SELECT (tab_)
    WAIT WIND "Compactare "+ALLTRIM(UPPER(tab_))+"..." NOWAIT
    PACK
    WAIT CLEAR

ENDFOR
SET DELETED ON                          && refacerea parametrului SET DELETED
CLOSE DATABASE                          && inchidem baza dedate
```

```
WAIT CLEAR                  && se sterge orice mesaj de pe ecran
RELEASE a
ON ERROR                    && se restaureaza mecanismul implicit de tratare a erorilor
```

Deşi procedura din listingul 17.3 va fi lansată de fiecare utilizator care părăseşte aplicaţia, ea se va executa în totalitate doar atunci când baza de date nu mai este utilizată pe alte calculatoare, deci când deschiderea exclusivă devine plauzibilă. S-a folosit instrucţiunea `WAIT WINDOW <mesaj>` pentru a afişa un mesaj într-o fereastră, în colţul din dreapta-sus al ecranului. În mod normal, această instrucţiune suspendă programul până la apăsarea unei taste, dar opţiunea `NOWAIT` împiedică acest lucru.

17.3.2. Compilarea proiectului

Compilarea unui proiect Visual FoxPro se realizează cu ajutorul butonului `Build` din `Project Manager` (figura 17.23)[81]. În fereastra `Build Options` putem selecta fie reconstruirea proiectului pentru reactualizarea legăturilor spre diversele fişiere componente, fie compilarea sa ca fişier executabil sau DLL. Aşa cum am văzut şi în capitolul 14, dacă proiectul conţine clase definite `OLE Public`, vom obţine componente COM de tip *in-process* sau *out-of-process*.

Este recomandabil să bifăm şi opţiunile `Recompile All Files` şi `Display Errors` pentru a fi siguri, pe de o parte, că ultimele modificări ale fişierelor componente sunt şi compilate, iar pe de altă parte, că orice eroare apărută la compilare va fi stocată într-un fişer de erori (extensia `.err`).

Observăm că, o dată instalat Visual Studio Service Pack 3 sau o versiune ulterioară, putem crea componente COM-DLL de tip *multifir* (*multi-threaded*) ce vor fi capabile să proceseze în paralel mai multe cereri de acces simultane.

Fişierul final, obţinut în urma compilării, va cuprinde doar acele componente pentru care a fost selectată opţiunea `Project→Include` din meniul sistem Visual FoxPro care se prezintă sub formă de buton cu două stări: `Include/Exclude`. Astfel, pentru componentele deja „incluse" vom regăsi în meniu opţiunea `Exclude` şi viceversa, pentru cele „excluse", opţiunea `Include`.

Notă. În afară de baza de date, toate celelalte componente (programe, rapoarte, formulare, clase etc.) sunt implicit incluse în proiect. Acelaşi regim îl au şi fişierele de imagini utilizate în formulare sau rapoarte.

Având în vedere aceste aspecte, utilizatorilor finali le vor fi livrate următoarele fişiere:

- executabilul sau componentele COM (pentru acestea din urmă sunt necesare şi fişierele asociate `.tlb` şi `.vbr`);
- fişierele bazei de date (`.dbc`, `.dct`, `.dcx`, `.dbf`, `.cdx`);
- fişierele de imagini ce nu sunt incluse în executabil;
- fişierul `help` compilat (`.chm`).

Aşadar, orice componentă „inclusă" se va regăsi în fişierul compilat şi nu va mai fi livrată utilizatorilor în forma ei independentă, în timp ce toate celelalte vor trebui să se regăsească pe discul utilizatorului.

81. Fereastra `Build Options` arată ca în figură numai dacă aveţi instalat Visual Studio Service Pack 3 sau versiunile ulterioare.

Notă. O tabelă poate fi şi ea inclusă în fişierul executabil final. În acest caz, fişierul corespunzător de pe disc (`.DBF`) poate să lipsească. Este însă important de reţinut faptul că tabela respectivă *nu va putea fi modificată, ci doar interogată.*

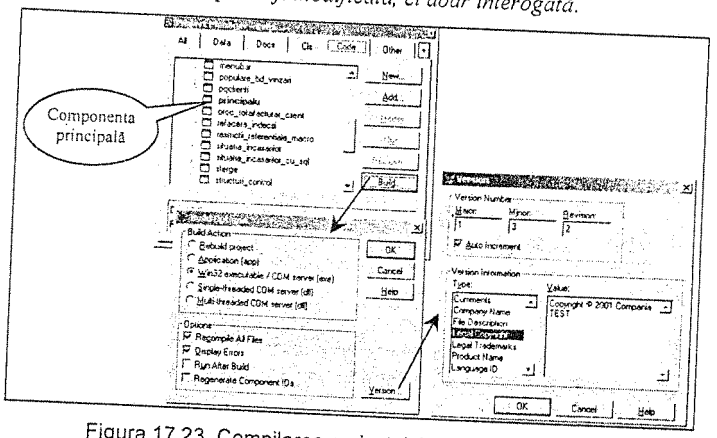


Figura 17.23. Compilarea proiectului în format executabil

Pentru un control cât mai bun al diverselor versiuni ale aplicaţiei, dispunem de butonul `Version` ce permite introducerea unui număr iniţial al versiunii cu posibilitatea de auto-incrementare automată ori de câte ori se recompilează proiectul. De asemenea, putem specifica diverse informaţii cu privire la copyright, numele companiei producătoare, numele aplicaţiei etc. Aceste informaţii vor fi încapsulate în fişierul executabil şi vor putea fi vizualizate cu ajutorul utilitarului Windows Explorer, opţiunea `File→Properties` (figura 17.24).

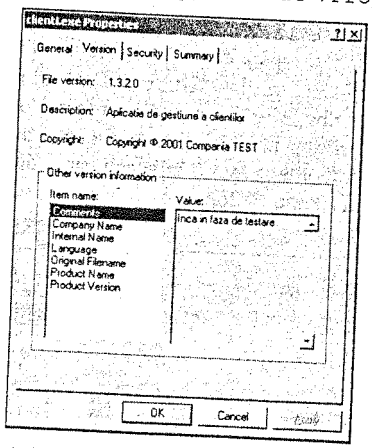


Figura 17.24. Informaţii legate de versiunea unui fişer executabil

Fişierul executabil obţinut în urma compilării va prelua implicit iconiţa specifică Visual FoxPro. Aceasta poate fi însă înlocuită dacă vom selecta opţiunea corespunzătoare,

`Attach Icon`, în fereastra `Project Information` din meniul `Project→ Project Info` (figura 17.25).

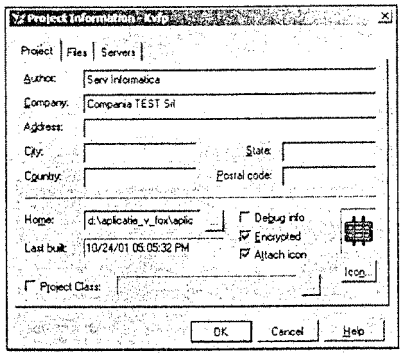


Figura 17.25. Fereastra `Project Information`

Atenție! Opțiunea `Encrypted` trebuie obligatoriu selectată. Altminteri fișierul obținut în urma compilării este suficient de inteligibil pentru o persoană avizată dacă este deschis cu un editor de texte (ex.: WordPad).

17.3.3. Crearea fișierelor de setup

După testarea și compilarea aplicației urmează instalarea acesteia pe calculatoarele utilizatorilor. În acest scop, simpla copiere a fișierelor componente pe discul destinație nu este suficientă. Pentru a funcționa, aplicația are nevoie de bibliotecile de bază ale Visual FoxPro.

O modalitate foarte simplă și rapidă de creare a kit-ului de instalare este furnizată prin intermediul opțiunii `Tools→Wizards→Setup` din meniul sistem. Astfel se pot crea, în mod asistat, fișierele de distribuție ce vor cuprinde o formă arhivată sau nu a modulelor aplicației, nucleul Visual FoxPro și modulul de instalare (`setup.exe`).

Primul pas, și cel mai important, constă în specificarea directorului în care se găsesc fișierele ce compun aplicația (figura 17.26).

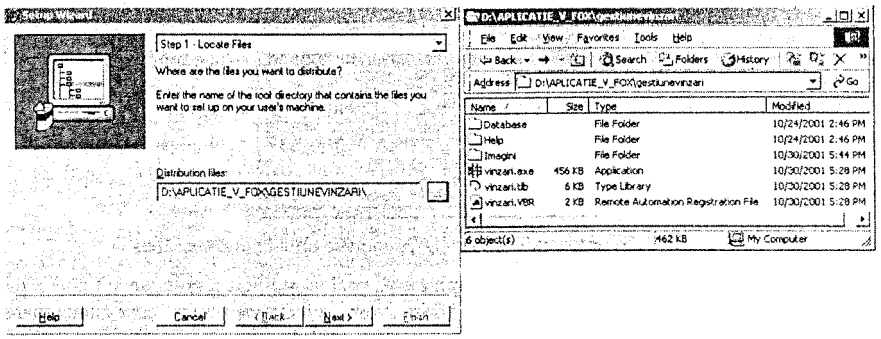


Figura 17.26. Pasul 1 – localizarea fișierelor aplicației

Directorul respectiv va conține doar fișierele strict necesare funcționării sistemului. Așa cum spuneam și în paragraful anterior, în acestă categorie intră: baza de date și tabelele acesteia, sistemul de ajutor (*help*), imaginile ce nu a fost incluse în proiectul compilat, fișierul executabil cu cele două tipuri asociate (`.tlb` și `.vbr`) în cazul în care acesta conține clase definite `OLE Public`.

În cel de-al doilea pas (figura 17.27) vom selecta componentele externe necesare astfel:

- `Visual FoxPro runtime` – vor fi incluse librăriile de bază ale Visual FoxPro. Această opțiune trebuie selectată obligatoriu;
- `Multi-threaded server runtime` – pentru funcționarea serverelor COM compilate în format DLL de tip multifir;
- `ActiveX controls` – deschide o listă cu toate componentele de tip `ActiveX` instalate pe calculatorul respectiv. Vom selecta din această listă doar pe acelea utilizate de aplicație;
- `COM components` – se vor selecta fișierele externe (`.exe` sau `.dll`) ce conțin biblioteci de componente COM utilizate de aplicație;
- `ODBC drivers` – în cazul în care aplicația se conectează la surse de date externe, altele decât Visual FoxPro (Oracle, SQL Server, Excel etc.);
- `Microsoft Graph 8.0 runtime` – dacă a fost utilizat acest utilitar pentru crearea graficelor.

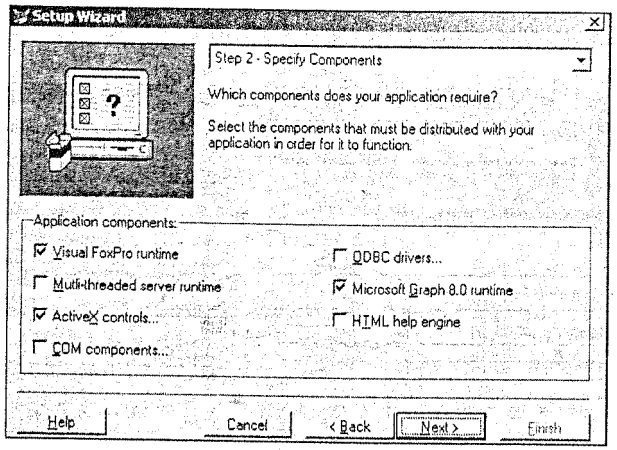


Figura 17.27. Pasul 2 – componente externe utilizate de aplicație

La pasul 3 se stabilește directorul în care vor fi create fișierele de `setup` și dacă se dorește arhivarea lor pentru distribuția prin intermediul disketelor ca mediu de stocare.

Pasul 4 constă în furnizarea informațiilor ce vor apărea pe fereastra principală de instalare (titlul aplicației și elemente de copyright).

La pasul 5 (figura 17.28) specificăm următoarele elemente referitoare la modul de instalare:

- numele și localizarea directorului-rădăcină al aplicației ce va fi creat de programul de setup;

- numele grupului de link-uri ce va fi creat în meniul principal al Windows-ului (`Start→Programs→Dengrup`);
- dacă utilizatorul va putea modifica localizarea directorului-rădăcină şi/sau numele grupului de link-uri.

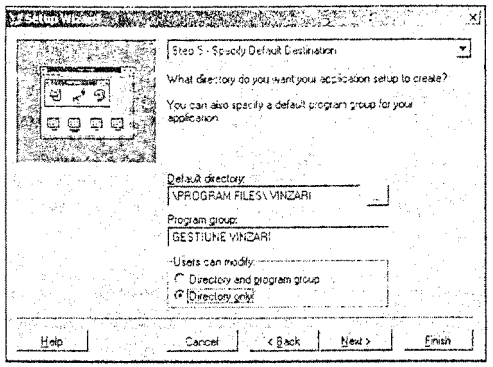


Figura 17.28. Pasul 5 – directorul-rădăcină al fişierelor aplicaţiei

La ultimul pas (figura 17.29) se vor selecta fişierele pentru care vor fi create link-uri în grupul de programe (din meniul principal al Windows-ului) cu numele precizat la pasul anterior.

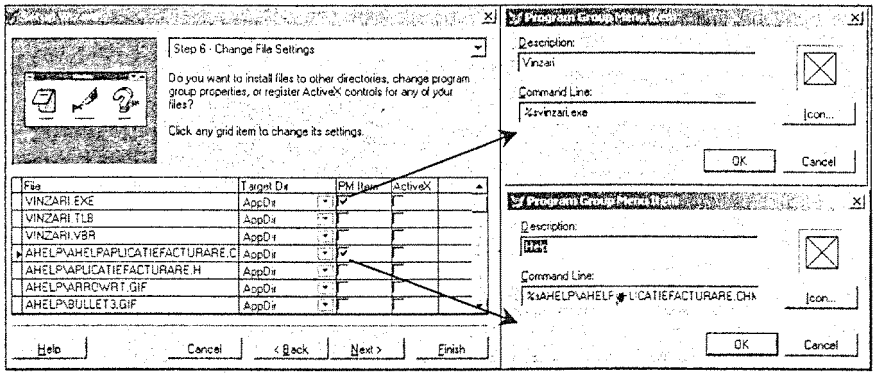


Figura 17.29. Pasul 6 – elementele grupului precizat în pasul anterior

După cum se observă şi în figura de mai sus, orice fişier al aplicaţiei poate fi inclus în grupul de programe.

În urma „bifării" căsuţei respective apare o fereastră cu următoarele zone:

- *description* – numele link-ului ce va fi creat în grupul de programe;
- *command line* – fişierul ce va fi lansat în execuţie ca urmare a unui click pe link-ul respectiv. Trebuie obligatoriu specificată şi calea de subdirectoare dacă fişierul nu se găseşte în rădăcina directorului principal al aplicaţiei. De asemenea, este **obligatorie** utilizarea secvenţei de caractere **%s** (cu literă mică) pentru a asigura independenţa faţă de localizarea aplicaţiei pe disc în momentul instalării.

După instalare, pentru exemplul dat în figura 17.29, grupul de programe din meniul Windows va arăta ca în figura 17.30.

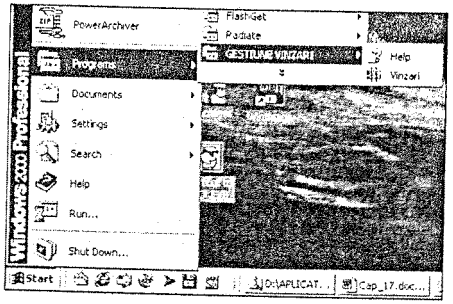


Figura 17.30. Grup de programe din meniul Windows

Bibliografie

Airinei, D., Filip, M., Grama, A., Fotache, M., Fînaru, L., Dumitriu, F., Țugui, A. – *Medii de programare*, Sedcom Libris, Iași, 2001

Bazian, M. *et al.* – *Totul despre Visual FoxPro*, Teora, București, 1999

Crețu, L. – „Aplicații Visual FoxPro pe Web", *Net Report*, noiembrie 2001

Fînaru, L. – *Programare vizuală și RAD*, Junimea, Iași, 2001

Fînaru, L., Brava, I. – *Visual Basic. Primii pași... și următorii*, Polirom, Iași, 2001

Fotache, M. – „Fiola de SQL (18). Exporturi FoxPro – Oracle/PostgreSQL", *PCReport&Byte*, nr. 109, octombrie 2000

Fotache, M. – „Fiola de SQL (20). Declanșatoare generalizate dedicate restricțiilor referențiale. Cazul Visual FoxPro 6", *PCReport&Byte*, nr. 111, decembrie 1999

Fotache, M. – *SQL. Dialecte DB2, Oracle și Visual FoxPro*, Polirom, Iași, 2001

Fotache, M., Strîmbei, C. – „Aplicații client-server Visual FoxPro – Oracle (I)", *PCReport&Byte*, nr. 94, iulie 2000

Fotache, M., Strîmbei, C. – „Aplicații client-server Visual FoxPro – Oracle (II)", *PCReport&Byte*, nr. 95, august 2000

Fotache, M., Strîmbei, C. – „De la Visual FoxPro la Oracle (I)", *PCReport&Byte*, nr. 87, decembrie 1999

Fotache, M., Strîmbei, C. – „De la Visual FoxPro la Oracle (II). Triggere pentru asigurarea restricțiilor referențiale", *PCReport&Byte*, nr. 88, ianuarie 2000

Fotache, M., Strîmbei, C. – „De la Visual FoxPro la Oracle (III). Triggere pentru asigurarea restricțiilor referențiale", *PCReport&Byte*, nr. 89, februarie 2000

Fotache, M., Strîmbei, C. – „De la Visual FoxPro la Oracle (IV). Tabele PL/SQL utilizate în declanșatoare", *PCReport&Byte*, nr. 90, martie 2000

Fotache, M., Strîmbei, C. – „Dezvoltarea aplicațiilor pentru rețea în VFP", *PCReport&Byte*, nr. 82, iulie 1999

Fotache, M., Strîmbei, C. – „Optimistic Row Buffering și tranzacții", *PCReport&Byte*, nr. 83, august 1999

Fotache, M., Strîmbei, C. – „Proceduri stocate și triggere în FoxPro", *PCReport&Byte*, nr. 80, mai 1999

Fotache, M., Strîmbei, C. – „Tabele derivate", *PCReport&Byte*, nr. 81, iunie 1999

Strahal, R. – Building applications with FoxISAPI, http://www.west-wind.com/

*** – www.msdn.microsoft.com;

BIBL. CENTR. UNIV.
„M. EMINESCU" IAȘI

CUPRINS

În loc de prefață 5

Conținutul lucrării 7

CAPITOLUL 1
VISUAL FOXPRO 11

1.1. Caracterizare generală 11
1.2. De la xBase la Visual FoxPro 12
1.3. Moduri de lucru în FoxPro 16
1.4. Crearea unui proiect Visual FoxPro 17
1.5. Configurarea mediului Visual FoxPro 20
1.5.1. Comenzile SET 21
1.5.2. Comenzile ON 25
1.5.3. Variabile de mediu 26
1.5.4. Fișierul CONFIG.FPW 27

CAPITOLUL 2
BAZE DE DATE ÎN VISUAL FOXPRO 30

2.1. Conceptul de „bază de date" și o confuzie de pe vremea xBase-urilor 30
2.2. Tabelele unei baze de date în Visual FoxPro 33
2.2.1. Tipuri de date în VFP 33
2.2.2. Conversia între tipurile de date 35
2.2.3. Valori nule 35
2.2.4. Crearea și modificarea structurii unei tabele 36
2.2.5. Restricțiile unei baze de date 39
2.2.6. Cheia primară 39
2.2.7. Restricția referențială 39
2.2.8. Alte tipuri de restricții ale bazei de date 40
2.3. Tabelele bazei de date VINZARI 40
2.4. Despre indecși 48
2.4.1. Crearea indecșilor în mod vizual 50
2.4.2. Crearea indecșilor prin comenzi 51
2.4.3. Probleme și erori legate de indecși 52
2.5. Restricțiile declarate într-o bază de date VFP 55
2.5.1. Definirea restricțiilor la nivel de atribut și înregistrare 55
2.5.2. Legăturile permanente dintre tabele și editarea restricțiilor referențiale . 58

2.5.3. Valori implicite pentru atributele de tip cheie străină 62
2.5.4. Momentul activării restricţiilor unei baze de date. 63

CAPITOLUL 3
NUCLEUL XBASE. ACTUALIZAREA ÎNREGISTRĂRILOR 65

3.1. Zone de lucru. Deschiderea tabelelor bazei de date sau a tabelelor libere 65
3.2. Noţiunea de pointer sau înregistrare curentă. Navigare 68
3.3. Acces la înregistrări în modul indexat şi neindexat 71
3.4. Adăugarea şi editarea înregistrărilor folosind modul asistat şi comenzi xBase 74
3.4.1. Editarea înregistrărilor în mod asistat 74
3.4.2. Comenzi xBase pentru actualizarea directă a înregistrărilor 79

CAPITOLUL 4
PROGRAMAREA ÎN VISUAL FOXPRO 82

4.1. Scurtă introducere în teoria programării 82
4.2. Structuri de control 83
4.3. Variabile şi constante. Instrucţiuni de atribuire în Visual FoxPro 86
4.4. Instrucţiuni de intrare-ieşire 91
4.5. Proceduri şi funcţii în Visual FoxPro 91
4.6. Macrosubstituţia : într-un cuvânt, mai multă putere 104
4.7. Importul şi exportul de date 106
4.7.1. Accesarea fişierelor binare în Visual FoxPro. Recuperarea datelor din tabele cu antetul (header-*ul*) *eronat.* 110
4.8. Tratarea erorilor în programele Visual FoxPro 113
4.9. Depanarea programelor cu instrumentul Debugger 119

CAPITOLUL 5
SQL ÎN VISUAL FOXPRO 126

5.1. Principalele comenzi SQL din VFP 126
5.2. Crearea şi modificarea structurii tabelelor 127
5.2.1. Crearea tabelelor şi declararea restricţiilor 127
5.2.2. Modificarea structurii tabelelor/restricţiilor în VFP 134
5.2.3. Ştergerea tabelelor 136
5.3. Comenzi de actualizare 136
5.3.1. Adăugarea unei linii 136
5.3.2. Ştergerea liniilor 137
5.3.3. Popularea bazei de date prin comenzi SQL 137
5.3.4. Modificarea valorilor unor atribute 141
5.4. Interogarea bazelor de date - fraza SELECT 142
5.4.1. Sintaxa de bază 142
5.4.2. Funcţii-agregat, cu şi fără grupare 145
5.4.3. Clauza TOP. Operatorii ALL şi ANY (SOME) 152
5.5. Actualizarea datelor folosind (sub)consultări 155
5.6. Includerea frazelor SELECT în programe 157
5.6.1. Salvarea rezultatului unei fraze SELECT în cursor 157
5.6.2. Utilizarea cursoarelor pentru actualizări 160
5.6.3. Alte consideraţii privind cursoarele 162
5.7. Variabile utilizate în consultări. Macrosubstituţie şi SQL 166
5.7.1. Variabile pentru filtrarea interogărilor 166
5.7.2. Macrosubstituţia şi fraze SELECT 167
5.7.3. Calculul automat al valorilor unor atribute agregate 169
5.7.4. Macrosubstituţie şi iteraţii SQL 170

CAPITOLUL 6
PROCEDURI STOCATE 174

6.1. Dicţionarul de date 174
6.1.1. Informaţii despre tabele 174
6.1.2. Informaţii despre atribute 176
6.1.3. Reflectarea în containerul bazei a relaţiilor permanente dintre tabele 178
6.2. Tipologia şi aria de utilizare a procedurilor stocate 180
6.3. Valori implicite obţinute prin funcţii stocate 182
6.4. Reguli de validare avansate 184
6.5. Declanşatoare 188
6.5.1. Declanşatoare pentru restricţii referenţiale 189
6.5.2. Atribute actualizate prin declanşatoare 190
6.5.3. Declanşatoare pentru controlul actualizărilor 193
6.6. Salvarea/restaurarea procedurilor stocate. Refacerea containerului BD 196
6.7. Declanşatoare generalizate pentru asigurarea restricţiilor referenţiale 200
6.8. Optimizarea procedurilor stocate pentru aplicaţii ce rulează pe un singur calculator (locale) 206
..... 216

CAPITOLUL 7
RAPOARTE ÎN VISUAL FOXPRO 224

7.1. Descrierea structurii unui raport 224
7.1.1. Structura de date 224
7.1.2. Structura de afişare 225
7.2. Definirea unui raport simplu 228
7.3. Rapoarte complexe 230
7.3.1. Imprimarea preferenţială pe anumite pagini 237
7.3.2. Gruparea înregistrărilor în raport 238
7.3.3. Variabilele de raport 239
7.3.4. Introducerea câmpurilor calculate 241
7.3.5. Formatul de afişare pentru datele numerice 242
7.3.6. Alinierea obiectelor în raport 244
..... 244

7.4. Setarea imprimantei la momentul execuţiei. Modalităţi de imprimare rapidă 246
7.5. Rapoarte dinamice în VFP 247

CAPITOLUL 8
FORMULARE VISUAL FOXPRO 253

8.1. Generalităţi. Tipuri de controale 253
8.2. Macheta primului formular. Tranzacţii 261
8.3. Proprietăţi, metode şi evenimente 266
8.3.1. Despre proprietăţi 267
8.3.2. Despre evenimente şi metode 271
8.4. Proprietăţi ale controalelor pentru formularul în studiu 274
8.5. Sursele de date ale controalelor. 282
8.5.1. Utilizarea `Data Environment` pentru deschiderea tabelelor 283
8.5.2. Deschiderea explicită a bazei de date şi tabelelor 285
8.6. Metodele formularului şi lansarea în execuţie 286

CAPITOLUL 9
TABELE VIRTUALE 292

9.1. Generalităţi despre tabele virtuale 292
9.1.1. Ce sunt tabelele virtuale şi care este utilitatea lor 292
9.1.2. Tabele virtuale în SQL-92 293
9.2. Crearea tabelelor derivate VFP în mod asistat 295
9.3. Conflicte la propagarea modificărilor în tabelele de bază 301
9.4. Obţinerea de informaţii privind structura tabelelor virtuale. Crearea *view*-urilor prin program 307
9.4.1. Tabelele derivate în dicţionarul de date 308
9.4.2. Crearea „programatică" a tabelelor virtuale 309
9.5. Alte consideraţii privind tabelele derivate în Visual FoxPro 312

CAPITOLUL 10
FORMULARE COMPLEXE 318

10.1. Controlul *Grid* şi controale pentru câmpuri de tip *General* 318
10.2. Tratarea erorilor în faza de execuţie a formularelor 328
10.3. Formulare şi tabele derivate. Specificul formularelor *Master-Detail* 330
10.4. Formulare parametrizate 339
10.5. Utilizarea controalelor ActiveX în formularele Visual FoxPro 340
10.5.1. Ce este şi ce-şi propune tehnologia ActiveX ? 340
10.5.2. Utilizarea controlului `Chart` într-un formular 341
10.5.3. Controlul `Calendar` sau o altă modalitate de lucru cu datele calendaristice 350
10.5.4. Controlul `ProgressBar` 355
10.5.5. Controalele `ImageList`, `TreeView` şi `ListView` 357
10.5.6. Controlul `Timer` 366
10.6. Depanarea formularelor cu utilitarul Debugger 368

CAPITOLUL 11
PROGRAMARE ORIENTATĂ-OBIECT ÎN VISUAL FOXPRO 370

11.1. Delimitări conceptuale 370
11.2. Clase în Visual FoxPro 370
11.2.1. De ce să folosim clase ? 374
11.2.2. Clasele de bază Visual FoxPro 374
11.2.3. Clasele definite de utilizator 374
11.2.4. Crearea de clase cu instrumentul `Class Designer` 375
11.2.5. Definirea şi instanţierea claselor prin cod 377
11.3. Gestionarea claselor Visual FoxPro 405
11.3.1. Instrumente „vizuale" pentru gestionarea claselor 410
11.3.2. Funcţii pentru gestionarea claselor 410
11.4. Crearea de biblioteci DLL cu Visual FoxPro 414
..... 417

CAPITOLUL 12
MENIURI ÎN VFP 420

12.1. Crearea meniurilor folosind generatorul de meniuri al Visual FoxPro 421
12.1.1. Planificarea meniului aplicaţiei 421
12.1.2. Crearea/definirea meniului principal şi submeniurilor corespunzătoare 422
12.1.3. Căi suplimentare de acces de la tastatură la opţiunile din meniu şi separarea opţiunilor în cadrul submeniurilor 424
12.1.4. Includerea în meniul aplicaţiei a unor submeniuri `system` 427
12.1.5. Dezactivarea opţiunilor dintr-un meniu 427
12.1.6. Asocierea directă a unor module de program - cod-sursă - la nivelul meniurilor 428
12.1.7. Generarea unui meniu şi câteva precizări sumare despre comenzile de definire a acestuia 433
12.2. Simularea unui meniu vertical şi crearea barelor de instrumente 435
12.2.1. Definirea unui meniu de pornire vertical 435
12.2.2. Crearea unei bare de instrumente pentru aplicaţie 445

CAPITOLUL 13
OPŢIUNI PENTRU DEZVOLTAREA APLICAŢIILOR ÎN REŢEA. ARHITECTURA *FILE-SERVER* 449

13.1. Arhitectura *file-server* 449
13.2. Deschiderea bazei de date şi tabelelor pentru lucru în reţea. Moduri de blocare 450
13.2.1. Deschiderea partajabilă a bazei de date şi tabelelor 451
13.2.2. Blocajul la nivel de tabelă 455

13.2.3. Blocajul antetului unei tabele 455
13.2.4. Blocajul la nivel de înregistrare 456
13.3. Despre buffering 458
13.3.1. Row Optimistic Buffering 459
13.3.2. Table Optimistic Buffering 460
13.4. Formular în care se foloseşte buffering optimist la nivel de înregistrare 461
13.5. Formular în care se foloseşte buffering optimist la nivel de tabelă 470

CAPITOLUL 14
APLICAŢII VISUAL FOXPRO PE WEB 477

14.1. FoxISAPI şi Servere COM - fundamentul aplicaţiilor VFP pe Web 477
14.1.1. FoxISAPI. Noţiune şi funcţionalitate 477
14.1.2. Componente COM în Visual FoxPro 479
14.1.3. Testarea şi depanarea componentelor COM 484
14.1.4. Utilizarea serverelor COM pe Web. Configurarea drepturilor de acces 485
14.2. Prima aplicaţie Visual FoxPro pe Web 488
14.3. Interogarea bazei de date. Interacţiunea cu utilizatorul 490
14.4. Formular de vizualizare/actualizare înregistrări 494
14.5. Managerul instanţelor multiple în FoxISAPI 506

CAPITOLUL 15
SECURITATE ÎN VISUAL FOXPRO 508

15.1. Citire/scriere Windows Registry. O modalitate de stocare a parolelor. 508
15.2. Obţinerea numelui utilizatorului din reţea 510
15.3. Restricţionarea editării tabelelor 513
15.4. Autorizarea editărilor anumitor linii 517

CAPITOLUL 16
APLICAŢII VFP CU SERVERE DE BAZE DE DATE 520

16.1. Specificul aplicaţiilor cu servere de baze de date 520
16.2. Crearea schemei de baze de date pe serverul Oracle 522
16.3. Configurarea mediului client-server : client VFP - server Oracle8i 535
16.3.1. ODBC, drivere ODBC 535
16.3.2. Clientul ORACLE8i şi protocolul de conectare NET8 536
16.3.3. Arhitectura funcţională a driverului Oracle ODBC Driver 537
16.4. Actualizarea tabelelor din baza de date gestionată de către serverul Oracle 541
16.4.1. Tabele derivate la distanţă 542
16.4.2. Tehnogia SQL Pass-Trough - *SPT* 544
16.4.3. Cursoare actualizabile 548
16.5. Formulare VFP - accesul şi modificarea datelor aflate pe serverul Oracle 549

CAPITOLUL 17
LIVRAREA PRODUSULUI FINIT 555

17.1. Crearea unui sistem de asistenţă. HTML Help Workshop 555
17.2. Cum se utilizează HTML Help Workshop ? 558
17.2.1. Crearea fişierelor de subiecte (conţinut) 559
17.2.2. Crearea unui cuprins (Table of Contents) 563
17.2.3. Crearea unui index 565
17.2.4. Ataşarea fişierului de help *compilat la o aplicaţie Visual FoxPro* 567
17.2.5. Asistenţă „senzitivă la context" 568
17.3. Construirea kit-ului de distribuţie 571
17.3.1. Noţiunea de „componentă principală" a unui proiect 571
17.3.2. Compilarea proiectului 574
17.3.3. Crearea fişierelor de setup 576

BIBLIOGRAFIE 581

BIBL. CENTR. UNIV.
„M. EMINESCU" IAŞI